ОСНОВЫ SQL И РАСШИРЕННЫЕ ВОЗМОЖНОСТИ

ВВЕДЕНИЕ

SQL – это стандартный язык для доступа к базам

данных. Как использовать SQL для доступа к

данным и управления ими в:

MySQL, SQL Server, Access, Oracle, Sybase, DB2 и других системах баз данных.

Синтаксис SQL:

```
SELECT Company, Country FROM Customers WHERE Country <> 'USA'
```

Результат SQL:

Компания	Страна
Island Trading	UK
Галерея гастронома	Испания
Винные погреба «Смеющийся Вакх»	Канада
Специалитеты Парижа	Франция
Бистро «Симонс»	Дания
Вольский Заезд	Польша

SQL – это «стандартный язык для доступа к базам данных и

управления ими».Что такое SQL?

- SQL расшифровывается как Structured Query Language (язык структурированных запросов)
- SQL позволяет получать доступ к базам данных и управлять ими
- SQL является стандартом ANSI (American National Standards Institute)

Что может делать SQL?

- SQL может выполнять запросы к базе данных
- SQL может извлекать данные из базы данных
- SQL может вставлять записи в базу данных
- SQL может обновлять записи в базе данных
- SQL может удалять записи из базы данных
- SQL может создавать новые базы данных
- SQL может создавать новые таблицы в базе данных
- SQL может создавать хранимые процедуры в базе данных
- SQL может создавать представления в базе данных
- SQL может устанавливать разрешения на таблицы, процедуры и представления

RDBMS

- RDBMS расшифровывается как Relational Database Management System (система управления реляционными базами данных).
- RDBMS является основой для SQL и для всех современных систем баз данных, таких как MS SQL Server, IBM DB2, Oracle, MySQL и Microsoft Access.
- Данные в RDBMS хранятся в объектах базы данных, называемых таблицами.
- Таблица представляет собой набор связанных записей данных и состоит из столбцов и строк.

ОСНОВЫ SQL

Таблицы базы данных

База данных чаще всего содержит одну или несколько таблиц. Каждая таблица идентифицируется по имени (например, «Клиенты» или «Заказы»). Таблицы содержат записи (строки) с данными. Ниже приведен пример таблицы с именем «Лица»:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер

Таблица выше содержит три записи (по одной для каждого человека) и пять столбцов (P_Id, LastName, FirstName, Address и City).

Операторы SQL

Большинство действий, которые необходимо выполнить с базой данных, выполняются с помощью операторов SQL. Следующий оператор SQL выберет все записи в таблице «Лица»:

```
SELECT * FROM Persons
```

Примечание:

- SQL не чувствителен к регистру
- Точка с запятой после операторов SQL?
- Некоторые системы баз данных требуют точку с запятой в конце каждого оператора SQL.
- Точка с запятой – это стандартный способ разделения каждого оператора SQL в системах баз данных, которые позволяют выполнять более одного оператора SQL в одном вызове к серверу.
- Мы используем MS Access и SQL Server 2000, и нам не нужно ставить точку с запятой после каждого

оператора SQL, но некоторые программы баз данных вынуждают вас использовать ее.

SQL DML и DDL

SQL можно разделить на две части: язык манипулирования данными(DML) иязык определения данных (DDL).

Команды запроса и обновления составляют**часть DML SQL**:
SELECT- извлекает данные из базы
данных**UPDATE**- обновляет данные в базе
данных**DELETE**- удаляет данные из базы
данных
INSERT INTO- вставляет новые данные в базу данных

Часть DDL SQLпозволяет создавать или удалять таблицы базы данных. Она также определяет индексы (ключи), задает связи между таблицами и накладывает ограничения между таблицами. Наиболее важные операторы DDL в SQL:

CREATE DATABASE- создает новую базу
данных**ALTER DATABASE**- изменяет базу
данных**CREATE TABLE**- создает новую
таблицу**ALTER TABLE**- изменяет таблицу
DROP TABLE- удаляет таблицу
CREATE INDEX- создает индекс (ключ поиска)
DROP INDEX- удаляет индекс

Оператор SQL SELECT

- Оператор SELECT используется для выбора данных из базы данных.
- Результат сохраняется в результирующей таблице, называемой результирующим набором.

- Синтаксис SQL SELECT

```
SELECT column_name(s)
FROM table_name
И
SELECT * FROM table_name
```

Примечание:SQL не чувствителен к регистру. SELECT – это то же самое, что и select.

Пример SQL-запроса SELECT

Таблица «Persons»:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер

Теперь мы хотим выбрать содержимое столбцов с именами «LastName» и «FirstName» из таблицы выше.

Мы используем следующий оператор SELECT:

SELECT LastName,FirstName FROM

Persons Результирующий набор будет

выглядеть следующим образом:

Фамилия	Имя
Хансен	Оля
Свендсон	Туве
Петтерсен	Кари

Пример SELECT *

Теперь мы хотим выбрать все столбцы из таблицы
"Persons". Мы используем следующий оператор SELECT:

```
SELECT * From Persons
```

Оператор SQL SELECT DISTINCT

- В таблице некоторые столбцы могут содержать повторяющиеся значения. Это не проблема, однако иногда вам нужно будет перечислить только разные (distinct) значения в таблице.
- Ключевое слово DISTINCT может использоваться для возврата только различных (distinct) значений.

- Синтаксис SQL SELECT DISTINCT

```
SELECT DISTINCT
имя_столбца(ов) FROM
имя_таблицы
```

SELECT DISTINCT Пример

Таблица «Persons»:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер

Теперь мы хотим выбрать только уникальные значения из столбца с именем «City» из таблицы выше. Мы используем следующий оператор SELECT:

SELECT DISTINCT Город FROM Люди

Результат будет выглядеть следующим

образом:

Город
Саннес
Ставангер

Предложение WHERE

- Предложение WHERE используется для фильтрации записей.
- Предложение WHERE используется для извлечения только тех записей, которые соответствуют заданному критерию.
- Синтаксис SQL WHERE

```
SELECT column_name(s)
FROM table_name
WHERE имя_столбца оператор значение
```

Пример предложения WHERE

Таблица «Persons»:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес

3	Петтерсен	Кари	Storgt 20	Ставангер

Теперь мы хотим выбрать из таблицы выше только тех людей, которые живут в городе «Санднес». Мы используем следующий оператор SELECT:

```
SELECT * FROM Люди
WHERE Город='Sandnes'
```

Результирующий набор будет выглядеть так:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес

Кавычки вокруг текстовых полей

SQL использует одинарные кавычки для текстовых значений (большинство систем баз данных также принимают двойные кавычки). Однако числовые значения не должны заключаться в кавычки.

Для текстовых значений:

Это верно:
SELECT * FROM Люди WHERE Имя='Tove' Это неправильно:
SELECT * FROM Люди WHERE Имя=Tove Для

числовых значений:

Это верно:
SELECT * FROM Люди WHERE Год=1965
Это неправильно:
SELECT * FROM Люди WHERE Год='1965'

Операторы, допустимые в предложении WHERE

С предложением WHERE можно использовать следующие операторы:

Оператор	Описание
=	Равно
<>	Не равно
>	Больше чем
<	Меньше чем
>=	Больше или равно
<=	Меньше или равно
МЕЖДУ	В пределах включительно диапазона
LIKE	Поиск по шаблону
IN	Если вы знаете точное значение, которое хотите получить, по крайней мере для одного из столбцов

Примечание:В некоторых версиях SQL оператор <> может быть записан как ! =

Операторы AND и OR

- Операторы AND и OR используются для фильтрации записей на основе более чем одного условия.
- Оператор AND отображает запись, если и первое, и второе условие истинны.
- Оператор OR отображает запись, если хотя бы одно из условий (первое или второе) истинно.

Пример оператора AND

Таблица «Persons»:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер

Теперь мы хотим выбрать только тех людей, у которых имя равно "Tove", А фамилия равна "Svendson":

Мы используем следующий оператор SELECT:

SELECT * FROM Persons
WHERE Имя='Tove' AND Фамилия='Svendson'

Результат будет выглядеть следующим образом:

P_Id	Фамилия	Имя	Адрес	Город
2	Свендсон	Туве	Borgvn 23	Саннес

Пример оператора OR

Теперь мы хотим выбрать только тех людей, у которых имя равно "Tove" ИЛИ имя равно "Ola":

Мы используем следующий оператор

SELECT: SELECT * FROM Persons
WHERE Имя='Tove' OR Имя='Ola'

Результирующий набор будет выглядеть так:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес

Комбинирование AND и OR

Вы также можете комбинировать AND и OR (используйте скобки для формирования сложных выражений).

Теперь мы хотим выбрать только тех людей, у которых фамилия равна "Svendson", И имя равно "Tove" ИЛИ "Ola":

Мы используем следующий оператор SELECT:

SELECT * FROM Persons
WHERE Фамилия='Svendson' AND (Имя='Tove' OR Имя='Ola') Результат

будет выглядеть следующим образом:

P_Id	Фамилия	Имя	Адрес	Город
2	Свендсон	Туве	Borgvn 23	Саннес

Ключевое слово ORDER BY

- Ключевое слово ORDER BY используется для сортировки результирующего набора.
- Ключевое слово ORDER BY используется для сортировки результирующего набора по указанному столбцу.
- Ключевое слово ORDER BY сортирует записи по возрастанию по умолчанию.
- Если вы хотите отсортировать записи по убыванию, вы можете использовать ключевое слово DESC.
- Синтаксис SQL ORDER BY

```
SELECT имя_столбца(ов)
FROM имя_таблицы ORDER BY имя_столбца(ов) ASC|DESC
```

Пример ORDER BY

Таблица «Persons»:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес

3	Петтерсен	Кари	Storgt 20	Ставангер
4	Нильсен	Том	Vingvn 23	Ставангер

Теперь мы хотим выбрать всех людей из таблицы выше, однако мы хотим отсортировать людей по их фамилии.

Мы используем следующую инструкцию

SELECT: SELECT * FROM Люди ORDER BY

Фамилия

Результирующий набор будет выглядеть так:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
4	Нильсен	Том	Vingvn 23	Ставангер
3	Петтерсен	Кари	Storgt 20	Ставангер
2	Свендсон	Туве	Borgvn 23	Саннес

Пример ORDER BY DESC

Теперь мы хотим выбрать всех людей из таблицы выше, однако мы хотим отсортировать людей по убыванию их фамилии.

Мы используем следующий оператор SELECT:

SELECT * FROM Люди ORDER BY Фамилия DESC

Результат будет выглядеть следующим образом:

P_Id	Фамилия	Имя	Адрес	Город
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер
4	Нильсен	Том	Vingvn 23	Ставангер
1	Хансен	Оля	Тимотеивн 10	Саннес

Оператор INSERT INTO

- Оператор INSERT INTO используется для вставки новых записей в таблицу.
- Оператор INSERT INTO используется для вставки новой строки в таблицу.
- Синтаксис SQL INSERT INTO
- Можно записать оператор INSERT INTO в двух формах.
- Первая форма не указывает имена столбцов, куда будут вставлены данные, только их

  значения: INSERT INTO имя_таблицы VALUES (значение1, значение2, значение3,...)

- Вторая форма указывает как имена столбцов, так и значения, которые будут вставлены:

  INSERT INTO имя_таблицы (столбец1, столбец2, столбец3,...) VALUES (значение1, значение2, значение3,...)

Пример SQL INSERT INTO

У нас есть следующая таблица "Persons":

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер

Теперь мы хотим вставить новую строку в таблицу

«Persons». Мы используем следующий оператор SQL:

INSERT INTO Люди VALUES (4,'Nilsen', 'Johan', 'Bakken 2', 'Stavanger')

Таблица "Люди" теперь будет выглядеть следующим образом:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер
4	Нильсен	Йохан	Баккен 2	Ставангер

Вставка данных только в указанные столбцы

Также можно добавлять данные только в определенные столбцы.

Следующая инструкция SQL добавит новую строку, но добавит данные только в столбцы "P_Id", "Фамилия" и "Имя":

INSERT INTO Люди (P_Id, Фамилия, Имя) VALUES (5, 'Tjessem', 'Jakob') Таблица

"Люди" теперь будет выглядеть следующим образом:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер
4	Нильсен	Йохан	Баккен 2	Ставангер
5	Tjessem	Якоб		

Оператор UPDATE

- Оператор UPDATE используется для обновления записей в таблице.
- Оператор UPDATE используется для обновления существующих записей в таблице.
- Синтаксис SQL UPDATE

```
UPDATE имя_таблицы
SET столбец1=значение,
столбец2=значение2,... WHERE
некоторый_столбец=некоторое_значение
```

Примечание:Обратите внимание на предложение WHERE в синтаксисе UPDATE. Предложение WHERE указывает, какая запись или записи должны быть обновлены. Если вы опустите предложение WHERE, будут обновлены все записи!

Пример SQL UPDATE

Таблица «Persons»:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер
4	Нильсен	Йохан	Баккен 2	Ставангер
5	Tjessem	Якоб		

Теперь мы хотим обновить данные о человеке «Tjessem, Jakob» в

таблице «Persons». Мы используем следующий оператор SQL:

```
ОБНОВИТЬ Persons
SET Адрес='Nissestien 67', Город='Sandnes'
WHERE LastName='Tjessem' AND FirstName='Jakob'
```

Таблица "Persons" теперь будет выглядеть так:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер
4	Нильсен	Йохан	Баккен 2	Ставангер
5	Tjessem	Якоб	Ниссетиен 67	Саннес

Предупреждение об операторе SQL UPDATE

Будьте осторожны при обновлении записей. Если бы мы опустили предложение WHERE в приведенном выше примере, например, так:

ОБНОВИТЬ Persons
SET Адрес='Nissestien 67', Город='Sandnes'

Таблица "Люди" выглядела бы следующим

образом:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Ниссетиен 67	Саннес
2	Свендсон	Туве	Ниссетиен 67	Саннес
3	Петтерсен	Кари	Ниссетиен 67	Саннес
4	Нильсен	Йохан	Ниссетиен 67	Саннес
5	Tjessem	Якоб	Ниссетиен 67	Саннес

Оператор DELETE

- Оператор DELETE используется для удаления записей в таблице.
- Оператор DELETE используется для удаления строк в таблице.
- Синтаксис SQL DELETE

```
DELETE FROM table_name
WHERE
some_column=some_value
```

Примечание:Обратите внимание на предложение WHERE в синтаксисе DELETE. Предложение WHERE указывает, какая запись или записи должны быть удалены. Если вы опустите предложение WHERE, все записи будут удалены!

Пример SQL DELETE

Таблица «Persons»:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер
4	Нильсен	Йохан	Баккен 2	Ставангер

5	Tjessem	Якоб	Ниссетиен 67	Саннес

Теперь мы хотим удалить человека «Tjessem, Jakob» из таблицы

«Persons». Мы используем следующий оператор SQL:

DELETE FROM Persons
WHERE LastName='Tjessem' AND FirstName='Jakob'

Таблица "Persons" теперь будет выглядеть так:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер
4	Нильсен	Йохан	Баккен 2	Ставангер

Удалить все строки

Можно удалить все строки в таблице, не удаляя саму таблицу. Это означает, что структура таблицы, атрибуты и индексы останутся нетронутыми:

DELETE FROM table_name

или

DELETE * FROM table_name

РАСШИРЕННЫЙ SQL

Предложение TOP

- Предложение TOP используется для указания количества возвращаемых записей.
- Предложение TOP может быть очень полезным для больших таблиц с тысячами записей. Возврат большого количества записей может повлиять на производительность.

Примечание:Не все системы баз данных поддерживают предложение TOP.

- Синтаксис SQL Server:

  SELECT TOP number|percent
  column_name(s) FROM table_name

- Эквивалент SQL SELECT TOP в MySQL и Oracle:

  - Синтаксис MySQL:

    SELECT column_name(s)
    FROM table_name LIMIT
    number

Пример:

```
SELECT *
FROM Persons
LIMIT 5
```

- Синтаксис Oracle

```
SELECT column_name(s)
FROM table_name
WHERE ROWNUM <= number
```

Пример

```
SELECT *
ИЗ Persons
WHERE ROWNUM <=5
```

Пример SQL TOP

Таблица «Persons»:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер
4	Нильсен	Том	Vingvn 23	Ставангер

Теперь мы хотим выбрать только две первые записи в таблице

выше. Мы используем следующий оператор SELECT:

SELECT TOP 2 * FROM Persons

Результирующий набор будет выглядеть так:

P_Id	Фамилия	Имя	Адрес	Город

1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес

Пример SQL TOP PERCENT

Таблица «Persons»:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер
4	Нильсен	Том	Vingvn 23	Ставангер

Теперь мы хотим выбрать только 50% записей в таблице выше.

Мы используем следующий оператор SELECT:

SELECT TOP 50 PERCENT * FROM Persons

Результирующий набор будет выглядеть так:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес

Подстановочные знаки SQL

- Подстановочные знаки SQL можно использовать при поиске данных в базе данных.
- Подстановочные знаки SQL могут заменять один или несколько символов при поиске данных в базе данных.

- Подстановочные знаки SQL должны использоваться с оператором SQL LIKE.
- В SQL можно использовать следующие подстановочные знаки:

Шаблон	Описание
%	Замена нуля или более символов
_	Замена ровно одного символа
[Список символов]	Любой одиночный символ из charlist
[^charlist] или [!список_символов]	Любой одиночный символ, не входящий в список charlist

Примеры подстановочных знаков SQL

У нас есть следующая таблица "Persons":

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер

Использование подстановочного знака %

Теперь мы хотим выбрать из таблицы "Persons" людей, проживающих в городе, название

которого начинается с "sa". Мы используем следующий оператор SELECT:

```
SELECT * FROM Persons
WHERE City LIKE 'sa%'
```

Результирующий набор будет выглядеть так:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес

Далее мы хотим выбрать из таблицы "Persons" людей, проживающих в городе, название которого содержит шаблон "nes".

Мы используем следующий оператор SELECT:

```
SELECT * FROM Persons
WHERE City LIKE '%nes%'
```

Результирующий набор будет выглядеть так:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес

Использование подстановочного знака _

Теперь мы хотим выбрать из таблицы "Persons" людей, имя которых начинается с любого символа, за которым следует "la".

Мы используем следующий оператор SELECT:

```
SELECT * FROM Persons
WHERE FirstName LIKE '_la'
```

Результирующий набор будет выглядеть так:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес

Далее мы хотим выбрать из таблицы "Persons" людей, фамилия которых начинается с "S", за которым следует любой символ, затем "end", затем любой символ и "on".

Мы используем следующий оператор SELECT:

```
SELECT * FROM Persons
WHERE LastName LIKE 'S_end_on'
```

Результат будет выглядеть

следующим образом:

P_Id	Фамилия	Имя	Адрес	Город
2	Свендсо н	Туве	Borgvn 23	Саннес

Использование подстановочного знака [charlist]

Теперь мы хотим выбрать из таблицы "Persons" людей, фамилия которых начинается с "b", "s" или "p".

Мы используем следующий оператор SELECT:

```
SELECT * FROM Persons
WHERE LastName LIKE '[bsp]%'
```

Результирующий набор будет выглядеть так:

P_Id	Фамилия	Имя	Адрес	Город
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер

Далее мы хотим выбрать из таблицы "Persons" людей, фамилия которых НЕ начинается с "b", "s" или "p".

Мы используем следующий оператор SELECT:

```
SELECT * FROM Persons
WHERE LastName LIKE '[!bsp]%'
```

Результирующий набор будет выглядеть так:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес

Оператор LIKE

- Оператор LIKE используется в предложении WHERE для поиска указанного шаблона в столбце.
- Оператор LIKE используется для поиска указанного шаблона в столбце.

Синтаксис SQL LIKE:

```
SELECT column_name(s)
FROM table_name
WHERE column_name LIKE pattern
```

Пример оператора LIKE

Таблица «Persons»:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер

Теперь мы хотим выбрать из таблицы выше людей, проживающих в городе,

название которого начинается с буквы «s». Мы используем следующий оператор

SELECT:

```
SELECT * FROM Persons
WHERE City LIKE 's%'
```

Знак "%" может использоваться для определения подстановочных знаков (пропущенных букв в шаблоне) как до, так и после шаблона.

Результирующий набор будет выглядеть так:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер

Далее, мы хотим выбрать из таблицы "Persons" людей, проживающих в городе, название

которого заканчивается на букву "s". Мы используем следующий оператор SELECT:

```
SELECT * FROM Persons
WHERE City LIKE '%s'
```

Результирующий набор будет выглядеть так:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес

Далее мы хотим выбрать из таблицы "Persons" людей, проживающих в городе, название которого содержит шаблон "tav".

Мы используем следующий оператор SELECT:

```
SELECT * FROM Persons
WHERE City LIKE '%tav%'
```

Результирующий набор будет выглядеть так:

P_Id	Фамилия	Имя	Адрес	Город
3	Петтерсен	Кари	Storgt 20	Ставангер

Также можно выбрать из таблицы "Persons" людей, проживающих в городе, название которого НЕ содержит шаблон "tav", используя ключевое слово NOT.

Мы используем следующий оператор SELECT:

```
SELECT * FROM Persons
WHERE City NOT LIKE '%tav%'
```

Результирующий набор будет выглядеть так:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес

Оператор IN

- Оператор IN позволяет указать несколько значений в предложении WHERE.

Синтаксис SQL IN:

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1,value2,...)
```

Пример оператора IN

Таблица «Persons»:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер

Теперь мы хотим выбрать из таблицы выше людей с фамилией "Hansen" или "Pettersen".

Мы используем следующий оператор

SELECT: SELECT * FROM Persons
WHERE LastName IN ('Hansen','Pettersen')

Результирующий набор будет выглядеть так:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер

Оператор BETWEEN

- Оператор BETWEEN используется в предложении WHERE для выбора диапазона данных между двумя значениями.
- Оператор BETWEEN выбирает диапазон данных между двумя значениями. Значениями могут быть числа, текст или даты.

- Синтаксис SQL BETWEEN:

```
SELECT column_name(s)
FROM table_name
WHERE column_name
BETWEEN value1 AND value2
```

Пример оператора BETWEEN

Таблица «Persons»:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер

Теперь мы хотим выбрать из таблицы выше людей с фамилией в алфавитном порядке между "Hansen" и "Pettersen".

Мы используем следующий оператор

SELECT: SELECT * FROM Persons
WHERE Фамилия
BETWEEN 'Hansen' AND 'Pettersen'

Результат будет выглядеть

следующим образом:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес

Примечание:

Оператор BETWEEN трактуется по-разному в разных базах данных.

В некоторых базах данных люди с фамилией "Hansen" или "Pettersen" не будут перечислены, потому что оператор BETWEEN выбирает только поля, которые находятся между тестовыми значениями и не включают их.

В других базах данных люди с фамилией "Hansen" или "Pettersen" будут перечислены, потому что оператор BETWEEN выбирает поля, которые находятся между тестовыми значениями и включают их.

А в других базах данных люди с фамилией "Hansen" будут перечислены, а "Pettersen" - нет (как в примере выше), потому что оператор BETWEEN выбирает поля между тестовыми значениями, включая первое тестовое значение и исключая последнее.

Поэтому: Проверьте, как ваша база данных обрабатывает оператор BETWEEN.

Пример 2

Чтобы отобразить людей вне диапазона в предыдущем примере, используйте NOT

BETWEEN: SELECT * FROM Persons
WHERE Фамилия
NOT BETWEEN 'Hansen' AND 'Pettersen'

Результат будет выглядеть следующим

образом:

P_Id	Фамилия	Имя	Адрес	Город
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер

Псевдоним SQL

- В SQL можно задать псевдоним для таблицы или столбца.
- Вы можете дать таблице или столбцу другое имя, используя псевдоним. Это может быть полезно, если у вас очень длинные или сложные имена таблиц или столбцов.

- Псевдоним может быть любым, но обычно он короткий.

- Синтаксис псевдонимов SQL для таблиц:

```
SELECT column_name(s)
FROM table_name
AS alias_name
```

- Синтаксис псевдонимов SQL для столбцов:

```
SELECT column_name AS alias_name
FROM table_name
```

Пример использования псевдонима

Предположим, у нас есть таблица с именем "Persons" и другая таблица с именем "Product_Orders". Мы дадим таблицам псевдонимы "p" и "po" соответственно.

Теперь мы хотим вывести список всех заказов, за которые отвечает

«Ola Hansen». Мы используем следующий оператор SELECT:

```
SELECT po.OrderID, p.LastName, p.FirstName
FROM Persons AS p, Product_Orders AS po
WHERE p.LastName='Hansen' AND p.FirstName='Ola'
```

Тот же оператор SELECT без псевдонимов:

```
SELECT Product_Orders.OrderID, Persons.LastName, Persons.FirstName
FROM Persons, Product_Orders
WHERE Persons.LastName='Hansen' AND Persons.FirstName='Ola'
```

Примечание:Как вы видите из двух операторов SELECT выше, псевдонимы могут упростить как написание, так и чтение запросов.

SQL JOIN

- SQL-соединения используются для запроса данных из двух или более таблиц на основе связи между определенными столбцами в этих таблицах.
- Ключевое слово JOIN используется в операторе SQL для запроса данных из двух или более таблиц на основе
  связи между определенными столбцами в этих таблицах.
- Таблицы в базе данных часто связаны друг с другом с помощью ключей.
- Первичный ключ - это столбец (или комбинация столбцов) с уникальным значением для каждой строки. Каждое значение первичного ключа должно быть уникальным в пределах таблицы. Цель состоит в том, чтобы связать данные между таблицами, не повторяя все данные в каждой таблице.

Посмотрите на таблицу «Persons»:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер

Обратите внимание, что столбец "P_Id" является первичным ключом в таблице "Persons". Это означает, что**никакие**две строки не могут иметь одинаковый P_Id. P_Id различает двух людей, даже если у них одинаковые имена.

Далее у нас есть таблица "Orders":

O_Id	НомерЗаказа	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

Обратите внимание, что столбец "O_Id" является первичным ключом в таблице "Orders", а столбец "P_Id" ссылается на людей в таблице "Persons" без использования их имен.

Обратите внимание, что связь между двумя таблицами выше - это столбец "P_Id".

Разные SQL JOIN

Прежде чем мы продолжим с примерами, мы перечислим типы JOIN, которые вы можете использовать, и различия между ними.

- **JOIN**: Возвращает строки, когда есть хотя бы одно совпадение в обеих таблицах
- **LEFT JOIN**: Возвращает все строки из левой таблицы, даже если в правой таблице нет совпадений
- **RIGHT JOIN**: Возвращает все строки из правой таблицы, даже если в левой таблице нет совпадений
- **FULL JOIN**: Возвращает строки, когда есть совпадение в одной из таблиц

Ключевое слово SQL INNER JOIN

Ключевое слово INNER JOIN возвращает строки, когда есть хотя бы одно совпадение в обеих таблицах.

Синтаксис SQL INNER JOIN:

```
SELECT column_name(s)
FROM table_name1
INNER JOIN table_name2
ПО table_name1.column_name=table_name2.column_name
```

PS:INNER JOIN то же самое, что и JOIN.

Пример ВНУТРЕННЕГО СОЕДИНЕНИЯ SQL

Таблица «Persons»:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер

Таблица «Orders»:

O_Id	НомерЗаказа	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

Теперь мы хотим вывести список всех людей,

имеющих какие-либо заказы. Мы используем

следующий оператор SELECT:

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons INNER JOIN Orders ON
Persons.P_Id=Orders.P_Id ORDER BY Persons.LastName
```

Результирующий набор будет выглядеть так:

Фамилия	Имя	НомерЗаказа
Хансен	Оля	22456
Хансен	Оля	24562
Петтерсен	Кари	77895

Петтерсен	Кари	44678

Ключевое слово INNER JOIN возвращает строки, когда есть хотя бы одно совпадение в обеих таблицах. Если в таблице "Persons" есть строки, которые не имеют совпадений в таблице "Orders", эти строки НЕ будут перечислены.

Ключевое слово SQL LEFT JOIN

- Ключевое слово LEFT JOIN возвращает все строки из левой таблицы (table_name1), даже если в правой таблице (table_name2) нет совпадений.

- Синтаксис SQL LEFT JOIN:

```
SELECT имя_столбца(ов)
FROM table_name1 LEFT JOIN table_name2
ПО table_name1.column_name=table_name2.column_name
```

- **Примечание:**В некоторых базах данных LEFT JOIN называется LEFT OUTER JOIN.

Пример LEFT JOIN в SQL

Таблица «Persons»:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер

Таблица «Orders»:

O_Id	НомерЗаказа	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

Теперь мы хотим вывести список всех людей и их заказов, если таковые имеются, из таблиц выше.

Мы используем следующий оператор SELECT:

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons LEFT JOIN Orders
ПО Persons.P_Id=Orders.P_Id
УПОРЯДОЧИТЬ ПО
Persons.LastName
```

Результирующий набор будет выглядеть так:

Фамилия	Имя	НомерЗаказа
Хансен	Оля	22456
Хансен	Оля	24562
Петтерсен	Кари	77895
Петтерсен	Кари	44678
Свендсон	Туве	

Примечания:Ключевое слово LEFT JOIN возвращает все строки из левой таблицы (Persons), даже если в правой таблице (Orders) нет совпадений.

Ключевое слово SQL RIGHT JOIN

- Ключевое слово RIGHT JOIN возвращает все строки из правой таблицы (table_name2), даже если в левой таблице (table_name1) нет совпадений.
- Синтаксис SQL RIGHT JOIN:

```
SELECT column_name(s)
FROM table_name1
RIGHT JOIN table_name2
ПО table_name1.column_name=table_name2.column_name
```

Примечание:В некоторых базах данных RIGHT JOIN называется RIGHT OUTER JOIN.

Пример RIGHT JOIN в SQL

Таблица «Persons»:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер

Таблица «Orders»:

O_Id	НомерЗаказа	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

Теперь мы хотим вывести список всех заказов, содержащих информацию о людях,

если таковые имеются, из таблиц выше. Мы используем следующий оператор

SELECT:

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo
FROM Persons
RIGHT JOIN Orders
ПО Persons.P_Id=Orders.P_Id
УПОРЯДОЧИТЬ ПО
Persons.LastName
```

Результирующий набор будет выглядеть так:

Фамилия	Имя	НомерЗаказа
Хансен	Оля	22456
Хансен	Оля	24562
Петтерсен	Кари	77895
Петтерсен	Кари	44678
		34764

Примечания:Ключевое слово RIGHT JOIN возвращает все строки из правой таблицы (Orders), даже если в левой таблице (Persons) нет совпадений.

Ключевое слово SQL FULL JOIN

- Ключевое слово FULL JOIN возвращает строки, когда есть совпадение в одной из таблиц.
- Синтаксис SQL FULL JOIN:

```
SELECT column_name(s)
FROM table_name1
FULL JOIN table_name2
ПО table_name1.column_name=table_name2.column_name
```

Пример полного объединения SQL

Таблица «Persons»:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер

Таблица «Orders»:

O_Id	НомерЗаказа	P_Id
1	77895	3
2	44678	3
3	22456	1
4	24562	1
5	34764	15

Теперь мы хотим вывести список всех людей и их заказов, а также всех заказов с соответствующими им людьми. Мы используем следующий оператор SELECT:

```
SELECT Persons.LastName, Persons.FirstName, Orders.OrderNo

FROM Persons
FULL JOIN Orders
ПО Persons.P_Id=Orders.P_Id
УПОРЯДОЧИТЬ ПО
Persons.LastName
```

Результирующий набор будет выглядеть так:

Фамилия	Имя	НомерЗаказа
Хансен	Оля	22456
Хансен	Оля	24562
Петтерсен	Кари	77895
Петтерсен	Кари	44678
Свендсон	Туве	
		34764

Примечания:Ключевое слово FULL JOIN возвращает все строки из левой таблицы (Persons) и все строки из правой таблицы (Orders). Если в таблице "Persons" есть строки, которые не имеют совпадений в таблице "Orders", или если в таблице "Orders" есть строки, которые не имеют совпадений в таблице "Persons", эти строки также будут перечислены.

Оператор UNION в SQL

 Оператор UNION используется для объединения результирующих наборов двух или более операторов SELECT.
 Обратите внимание, что каждый оператор SELECT в UNION должен иметь одинаковое количество столбцов. Столбцы также должны иметь схожие типы данных. Кроме того, столбцы в каждом операторе SELECT должны быть в одном и том же порядке.

 Синтаксис SQL UNION:

```
SELECT column_name(s) FROM table_name1
UNION
SELECT имя_столбца(ов) FROM имя_таблицы2
```

Примечание:По умолчанию оператор UNION выбирает только уникальные значения. Чтобы разрешить дубликаты, используйте UNION ALL.

 Синтаксис SQL UNION ALL:

```
SELECT column_name(s) FROM
table_name1 UNION ALL
SELECT имя_столбца(ов) FROM имя_таблицы2
```

Примечание:Имена столбцов в результирующем наборе UNION всегда равны именам столбцов в первом операторе SELECT в UNION.

Пример SQL UNION

Посмотрите на следующие таблицы:

"Employees_Norway":

E_ID	Имя_Сотрудника
01	Хансен, Ола
02	Свендсон, Туве
03	Свендсон, Стивен
04	Петтерсен, Кари

"Employees_USA":

E_ID	Имя_Сотрудника
01	Тернер, Салли

02	Кент, Кларк
03	Свендсон, Стивен
04	Скотт, Стивен

Теперь мы хотим вывести список**всех различных**сотрудников в

Норвегии и США. Мы используем следующий оператор SELECT:

```
SELECT E_Name FROM Employees_Norway
UNION
SELECT E_Name FROM Employees_USA
```

Результат будет выглядеть следующим

образом:

Имя_Сотрудника
Хансен, Ола
Свендсон, Туве
Свендсон, Стивен
Петтерсен, Кари
Тернер, Салли
Кент, Кларк
Скотт, Стивен

Примечание:Эта команда не может быть использована для вывода списка всех сотрудников в Норвегии и США. В приведенном выше примере у нас есть два сотрудника с одинаковыми именами, и только один из них будет указан в списке. Команда UNION выбирает только уникальные значения.

Пример SQL UNION ALL

Теперь мы хотим вывести список**всех**сотрудников в

Норвегии и США: SELECT E_Name FROM

```
Employees_Norway
UNION ALL
SELECT E_Name FROM Employees_USA
```

Результат

Имя_Сотрудника
Хансен, Ола
Свендсон, Туве
Свендсон, Стивен
Петтерсен, Кари
Тернер, Салли
Кент, Кларк
Свендсон, Стивен
Скотт, Стивен

Оператор SQL SELECT INTO

- Оператор SQL SELECT INTO может быть использован для создания резервных копий таблиц.
- Оператор SELECT INTO выбирает данные из одной таблицы и вставляет их в другую таблицу.

- Оператор SELECT INTO чаще всего используется для создания резервных копий таблиц.

- Синтаксис SQL SELECT INTO

Мы можем выбрать все столбцы в новую

таблицу: SELECT *
В new_table_name [В externaldatabase] ИЗ
old_tablename

Или мы можем выбрать только нужные нам столбцы в

новую таблицу: SELECT column_name(s)
В new_table_name [В externaldatabase] ИЗ
old_tablename

Пример SQL SELECT INTO

Создайте резервную копию- Теперь мы хотим создать точную копию данных в нашей

таблице "Persons". Мы используем следующий оператор SQL:

```
SELECT *
INTO Persons_Backup
FROM Persons
```

Мы также можем использовать предложение IN для копирования таблицы в другую базу данных:

```
SELECT *
INTO Persons_Backup IN 'Backup.mdb'
FROM Persons
```

Мы также можем скопировать только несколько полей в новую таблицу:

```
SELECT LastName,FirstName
INTO Persons_Backup
ИЗ Persons
```

SQL SELECT INTO - С предложением WHERE

Мы также можем добавить предложение WHERE.

Следующий оператор SQL создает таблицу "Persons_Backup" только с теми лицами, которые живут в городе "Sandnes":

```
SELECT LastName,Firstname
INTO Persons_Backup FROM
Persons
WHERE City='Sandnes'
```

SQL SELECT INTO - Объединение таблиц

Выбор данных из нескольких таблиц также возможен.

В следующем примере создается таблица "Persons_Order_Backup", содержащая данные из двух таблиц "Persons" и "Orders":

```
SELECT Persons.LastName,Orders.OrderNo
INTO Persons_Order_Backup
FROM Persons
INNER JOIN Orders
ON Persons.P_Id=Orders.P_Id
```

Оператор CREATE DATABASE

- Оператор CREATE DATABASE используется для создания базы данных.
- Синтаксис SQL CREATE DATABASE:

```
CREATE DATABASE database_name
```

СОЗДАТЬ БАЗУ ДАННЫХ Пример

Теперь мы хотим создать базу данных с именем "my_db".

Мы используем следующий оператор CREATE DATABASE:

```
CREATE DATABASE my_db
```

Таблицы базы данных можно добавить с помощью оператора

CREATE TABLE. Оператор CREATE TABLE

Оператор CREATE TABLE используется для создания таблицы в базе данных.

Синтаксис SQL CREATE TABLE:

```
CREATE TABLE
table_name (
column_name1
data_type,
column_name2
data_type,
column_name3
data_type,
....
)
```

Тип данных указывает, какой тип данных может содержать столбец. Для полного обзора всех типов данных, доступных в MS Access, MySQL и SQL Server.

CREATE TABLE Пример

Теперь мы хотим создать таблицу с именем "Persons", которая содержит пять столбцов: P_Id,

LastName, FirstName, Address и City.

Мы используем следующую команду CREATE TABLE:

```
CREATE TABLE Persons
(
P_Id int,
LastName varchar(255),
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

Столбец P_Id имеет тип int и будет содержать число. Столбцы LastName, FirstName, Address и City имеют тип varchar с максимальной длиной 255 символов.

Пустая таблица "Persons" теперь будет выглядеть так:

P_Id	Фамилия	Имя	Адрес	Город

Пустую таблицу можно заполнить данными с помощью оператора INSERT INTO.

Ограничения SQL

- Ограничения используются для ограничения типа данных, которые могут быть помещены в таблицу.
- Ограничения могут быть указаны при создании таблицы (с помощью оператора CREATE TABLE) или после создания таблицы (с помощью оператора ALTER TABLE).

Мы сосредоточимся на следующих ограничениях:

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT

В следующих главах будет подробно описано каждое ограничение.

Ограничение SQL NOT NULL

- Ограничение NOT NULL предписывает столбцу НЕ принимать значения NULL.
- Ограничение NOT NULL предписывает полю всегда содержать значение. Это означает, что вы не можете вставить новую запись или обновить запись, не добавив значение в это поле.

Следующий SQL предписывает столбцу "P_Id" и столбцу "LastName" не принимать значения NULL:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Адрес varchar(255),
Город varchar(255)
)
```

Ограничение SQL UNIQUE

- Ограничение UNIQUE однозначно идентифицирует каждую запись в таблице базы данных.
- Ограничения UNIQUE и PRIMARY KEY оба гарантируют уникальность для столбца или набора столбцов.
- Ограничение PRIMARY KEY автоматически имеет определенное на нем ограничение UNIQUE.
- Обратите внимание, что в таблице может быть много ограничений UNIQUE, но только одно ограничение PRIMARY KEY.

Ограничение SQL UNIQUE в CREATE TABLE

Следующий SQL создает ограничение UNIQUE для столбца "P_Id" при создании таблицы "Persons":

MySQL:

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

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons
(
P_Id int NOT NULL UNIQUE,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Адрес varchar(255),
Город varchar(255)
)
```

Чтобы задать имя ограничению UNIQUE и определить ограничение UNIQUE для нескольких столбцов, используйте следующий синтаксис SQL:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Адрес varchar(255),
Город varchar(255),
CONSTRAINT uc_PersonID UNIQUE (P_Id,LastName)
)
```

Ограничение SQL UNIQUE в ALTER TABLE

Чтобы создать ограничение UNIQUE для столбца "P_Id", когда таблица уже создана, используйте следующий SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD UNIQUE (P_Id)
```

Чтобы задать имя ограничению UNIQUE и определить ограничение UNIQUE для нескольких столбцов, используйте следующий синтаксис SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ИЗМЕНИТЬ ТАБЛИЦУ Persons
ADD CONSTRAINT uc_PersonID UNIQUE (P_Id,LastName)
```

Чтобы удалить ограничение UNIQUE

Чтобы удалить ограничение UNIQUE, используйте следующий SQL:

MySQL:

```
ALTER TABLE Persons
DROP INDEX uc_PersonID
```

SQL Server / Oracle / MS Access:

ИЗМЕНИТЬ ТАБЛИЦУ Persons
DROP CONSTRAINT uc_PersonID

Ограничение SQL PRIMARY KEY

- Ограничение PRIMARY KEY однозначно идентифицирует каждую запись в таблице базы данных.
- Первичные ключи должны содержать уникальные значения.
- Столбец первичного ключа не может содержать значения NULL.
- Каждая таблица должна иметь первичный ключ, и каждая таблица может иметь только ОДИН первичный ключ.

Ограничение SQL PRIMARY KEY в CREATE TABLE

Следующий SQL создает PRIMARY KEY для столбца "P_Id" при создании таблицы "Persons":

MySQL:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Адрес varchar(255),
Город varchar(255),
Первичный ключ (P_Id)
)
```

SQL Server / Oracle / MS Access:

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

Чтобы задать имя ограничению PRIMARY KEY и определить ограничение PRIMARY KEY для нескольких столбцов, используйте следующий синтаксис SQL:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Адрес varchar(255),
Город varchar(255),
CONSTRAINT pk_PersonID PRIMARY KEY (P_Id,LastName)
)
```

Ограничение SQL PRIMARY KEY в ALTER TABLE

Чтобы создать ограничение PRIMARY KEY для столбца "P_Id", когда таблица уже создана, используйте следующий SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER TABLE Persons
ADD PRIMARY KEY
(P_Id)
```

Чтобы задать имя ограничению PRIMARY KEY и определить ограничение PRIMARY KEY для нескольких столбцов, используйте следующий синтаксис SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ИЗМЕНИТЬ ТАБЛИЦУ Persons
ADD CONSTRAINT pk_PersonID PRIMARY KEY (P_Id,LastName)
```

Примечание:Если вы используете оператор ALTER TABLE для добавления первичного ключа, столбцы первичного ключа должны быть уже объявлены как не содержащие значения NULL (при первом создании таблицы).

Чтобы удалить ограничение PRIMARY KEY

Чтобы удалить ограничение PRIMARY KEY, используйте следующий SQL:

MySQL:

```
ALTER TABLE Persons
DROP PRIMARY KEY
```

SQL Server / Oracle / MS Access:

```
ИЗМЕНИТЬ ТАБЛИЦУ Persons
DROP CONSTRAINT pk_PersonID
```

Ограничение SQL FOREIGN KEY

- FOREIGN KEY в одной таблице указывает на PRIMARY KEY в другой таблице.

Давайте проиллюстрируем внешний ключ на примере. Посмотрите на

следующие две таблицы: Таблица "Persons":

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер

Таблица «Orders»:

O_Id	НомерЗаказа	P_Id
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Обратите внимание, что столбец "P_Id" в таблице "Orders" указывает на столбец "P_Id" в

таблице "Persons". Столбец "P_Id" в таблице "Persons" является PRIMARY KEY в таблице

"Persons".

Столбец "P_Id" в таблице "Orders" является FOREIGN KEY в таблице "Orders".

Ограничение FOREIGN KEY используется для предотвращения действий, которые могут разрушить связи между таблицами.

Ограничение FOREIGN KEY также предотвращает вставку неверных данных в столбец внешнего ключа, поскольку оно должно быть одним из значений, содержащихся в таблице, на которую оно указывает.

Ограничение SQL FOREIGN KEY в CREATE TABLE

Следующий SQL создает FOREIGN KEY для столбца "P_Id" при создании таблицы "Orders":

MySQL:

```
CREATE TABLE
Заказы (
O_Id int NOT NULL,
OrderNo int NOT NULL,
```

```
P_Id int,
ПЕРВИЧНЫЙ КЛЮЧ (O_Id),
FOREIGN KEY (P_Id) REFERENCES Persons(P_Id)
)
```

SQL Server / Oracle / MS Access:

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

Чтобы задать имя ограничению FOREIGN KEY и определить ограничение FOREIGN KEY для нескольких столбцов, используйте следующий синтаксис SQL:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE
Заказы (
O_Id int NOT NULL,
OrderNo int NOT NULL,
P_Id int,
ПЕРВИЧНЫЙ КЛЮЧ (O_Id),
CONSTRAINT fk_PerOrders FOREIGN KEY (P_Id)
REFERENCES Persons(P_Id)
)
```

Ограничение SQL FOREIGN KEY в ALTER TABLE

Чтобы создать ограничение FOREIGN KEY для столбца "P_Id", когда таблица "Orders" уже создана, используйте следующий SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ИЗМЕНИТЬ ТАБЛИЦУ
Заказы ДОБАВИТЬ
ВНЕШНИЙ КЛЮЧ (P_Id)
ССЫЛКИ Persons(P_Id)
```

Чтобы задать имя ограничению FOREIGN KEY и определить ограничение FOREIGN KEY для нескольких столбцов, используйте следующий синтаксис SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ALTER                    ТАБЛИЦА          Заказы
ДОБАВИТЬ                 ОГРАНИЧЕНИЕ      fk_PerOrders
FOREIGN                  КЛЮЧ             (P_Id)
ССЫЛКИ Persons(P_Id)
```

Чтобы УДАЛИТЬ ограничение ВНЕШНЕГО КЛЮЧА

Чтобы удалить ограничение ВНЕШНЕГО КЛЮЧА, используйте следующий SQL-запрос:

MySQL:

ИЗМЕНИТЬ ТАБЛИЦУ Заказы
УДАЛИТЬ ВНЕШНИЙ КЛЮЧ

fk_PerOrders SQL Server / Oracle / MS

Access:

ИЗМЕНИТЬ ТАБЛИЦУ Заказы
УДАЛИТЬ ОГРАНИЧЕНИЕ fk_PerOrders

Ограничение SQL CHECK

- Ограничение CHECK используется для ограничения диапазона значений, которые могут быть помещены в столбец.
- Если вы определяете ограничение CHECK для одного столбца, оно допускает только определенные значения для этого столбца.
- Если вы определяете ограничение CHECK для таблицы, оно может ограничивать значения в определенных столбцах на основе значений в других столбцах в строке.

Ограничение SQL CHECK при создании таблицы (CREATE TABLE)

Следующий SQL-запрос создает ограничение CHECK для столбца "P_Id" при создании таблицы "Persons". Ограничение CHECK указывает, что столбец "P_Id" должен содержать только целые числа больше 0.

My SQL:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
CHECK (P_Id>0)
)
```

SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons
(
P_Id int NOT NULL CHECK (P_Id>0),
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Адрес varchar(255),
```

```
Город varchar(255)
)
```

Чтобы задать имя ограничению CHECK и определить ограничение CHECK для нескольких столбцов, используйте следующий синтаксис SQL:

MySQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Адрес varchar(255),
Город varchar(255),
CONSTRAINT chk_Person CHECK (P_Id>0 AND City='Sandnes')
)
```

Ограничение SQL CHECK при изменении таблицы (ALTER TABLE)

Чтобы создать ограничение CHECK для столбца "P_Id", когда таблица уже создана, используйте следующий SQL-запрос:

MySQL / SQL Server / Oracle / MS Access:

```
ИЗМЕНИТЬ
ТАБЛИЦУ Persons
ДОБАВИТЬ CHECK
(P_Id>0)
```

Чтобы задать имя ограничению CHECK и определить ограничение CHECK для нескольких столбцов, используйте следующий синтаксис SQL:

MySQL / SQL Server / Oracle / MS Access:

```
ИЗМЕНИТЬ ТАБЛИЦУ Persons
ДОБАВИТЬ ОГРАНИЧЕНИЕ chk_Person CHECK (P_Id>0 AND City='Sandnes')
```

Чтобы УДАЛИТЬ ограничение CHECK

Чтобы удалить ограничение CHECK, используйте следующий SQL-запрос:

SQL Server / Oracle / MS Access:

```
ИЗМЕНИТЬ ТАБЛИЦУ Persons
УДАЛИТЬ ОГРАНИЧЕНИЕ chk_Person
```

Ограничение SQL DEFAULT

- Ограничение DEFAULT используется для вставки значения по умолчанию в столбец.

 Значение по умолчанию будет добавлено ко всем новым записям, если не указано другое значение.

Ограничение SQL DEFAULT при создании таблицы (CREATE TABLE)

Следующий SQL-запрос создает ограничение DEFAULT для столбца "City" при создании таблицы "Persons":

My SQL / SQL Server / Oracle / MS Access:

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255) DEFAULT 'Sandnes'
)
```

Ограничение DEFAULT также может использоваться для вставки системных значений с помощью функций, таких как GETDATE():

```
CREATE TABLE
Заказы (
O_Id int NOT NULL,
OrderNo int NOT NULL,
P_Id int,
OrderDate date DEFAULT GETDATE()
)
```

Ограничение SQL DEFAULT при изменении таблицы (ALTER TABLE)

Чтобы создать ограничение DEFAULT для столбца "City", когда таблица уже создана, используйте следующий SQL-запрос:

MySQL:

```
ALTER ТАБЛ Persons
ИЗМЕНИТЬ City УСТАНОВИТЬ ПО
```

SQL Server / Oracle / MS Access:

```
ИЗМЕНИТЬ ТАБЛИЦУ Persons
ИЗМЕНИТЬ СТОЛБЕЦ City УСТАНОВИТЬ ПО УМОЛЧАНИЮ 'SANDNES'
```

Чтобы УДАЛИТЬ ограничение DEFAULT

Чтобы удалить ограничение DEFAULT, используйте следующий SQL-запрос:

MySQL:

ИЗМЕНИТЬ ТАБЛИЦУ
Persons ИЗМЕНИТЬ City
УДАЛИТЬ ПО
УМОЛЧАНИЮ

SQL Server / Oracle / MS Access:

ИЗМЕНИТЬ ТАБЛИЦУ Persons
ИЗМЕНИТЬ СТОЛБЕЦ City УДАЛИТЬ ПО УМОЛЧАНИЮ

Индексы

- Оператор CREATE INDEX используется для создания индексов в таблицах.
- Индексы позволяют приложению базы данных быстро находить данные, не читая всю таблицу.
- Индекс может быть создан в таблице для более быстрого и эффективного поиска данных.
- Пользователи не видят индексы, они используются только для ускорения поиска/запросов.

Примечание:Обновление таблицы с индексами занимает больше времени, чем обновление таблицы без индексов (потому что индексы также нуждаются в обновлении). Поэтому следует создавать индексы только для столбцов (и таблиц), по которым будет часто выполняться поиск.

Синтаксис SQL CREATE INDEX

Создает индекс в таблице. Дубликаты значений разрешены:

CREATE INDEX имя_индекса
ON имя_таблицы
(имя_столбца)

Синтаксис SQL CREATE UNIQUE INDEX

Создает уникальный индекс в таблице. Дубликаты значений не допускаются:

CREATE UNIQUE INDEX
имя_индекса ON имя_таблицы
(имя_столбца)

Примечание:Синтаксис создания индексов различается в разных базах данных. Поэтому: проверьте синтаксис создания индексов в вашей базе данных.

СОЗДАТЬ ИНДЕКС (CREATE INDEX) Пример

Приведенный ниже SQL-оператор создает индекс с именем "PIndex" для столбца "LastName" в таблице "Persons":

```
СОЗДАТЬ ИНДЕКС PIndex
ON Persons (LastName)
```

Если вы хотите создать индекс по комбинации столбцов, вы можете перечислить имена столбцов в скобках, разделяя их запятыми:

СОЗДАТЬ ИНДЕКС PIndex
ON Persons (LastName, FirstName)

Оператор DROP INDEX

- Индексы, таблицы и базы данных можно легко удалить с помощью оператора DROP.
- Оператор DROP INDEX используется для удаления индекса в таблице.

Синтаксис DROP INDEX для MS Access:
DROP INDEX имя_индекса ON имя_таблицы

Синтаксис DROP INDEX для MS SQL Server:
DROP INDEX имя_таблицы.имя_индекса

Синтаксис DROP INDEX для DB2/Oracle:
DROP INDEX имя_индекса

Синтаксис DROP INDEX для MySQL:
ALTER TABLE имя_таблицы DROP INDEX имя_индекса

Оператор DROP TABLE

Оператор DROP TABLE используется для

удаления таблицы. DROP TABLE имя_таблицы

Оператор DROP DATABASE

Оператор DROP DATABASE используется для удаления

базы данных. DROP DATABASE имя_базы_данных

Оператор TRUNCATE TABLE

Что делать, если мы хотим удалить только данные внутри таблицы, а не

саму таблицу? В этом случае используйте оператор TRUNCATE TABLE:

TRUNCATE TABLE имя_таблицы

Оператор ALTER TABLE

Оператор ALTER TABLE используется для добавления, удаления или изменения

столбцов в существующей таблице. Синтаксис SQL ALTER TABLE

Чтобы добавить столбец в таблицу, используйте следующий синтаксис:

```
ALTER TABLE
имя_таблицы ADD
имя_столбца тип_данных
```

Чтобы удалить столбец из таблицы, используйте следующий синтаксис (обратите внимание, что некоторые системы баз данных не позволяют удалять столбцы):

```
ALTER TABLE имя_таблицы
DROP COLUMN имя_столбца
```

Чтобы изменить тип данных столбца в таблице, используйте следующий синтаксис:

```
ALTER TABLE имя_таблицы
ALTER COLUMN имя_столбца тип_данных
```

Пример SQL ALTER TABLE

Посмотрите на таблицу «Persons»:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер

Теперь мы хотим добавить столбец с именем «DateOfBirth» в таблицу

«Persons». Мы используем следующий оператор SQL:

```
ALTER TABLE Persons
ADD DateOfBirth date
```

Обратите внимание, что новый столбец «DateOfBirth» имеет тип date и будет содержать дату. Тип данных указывает, какой тип данных может содержать столбец. Для полного обзора всех типов данных, доступных в MS Access, MySQL и SQL Server, перейдите к нашей полной ссылке на типы данных.

Таблица "Persons" теперь будет выглядеть так:

P_Id	Фамилия	Имя	Адрес	Город	ДатаРождения
1	Хансен	Оля	Тимотеивн 10	Саннес	
2	Свендсон	Туве	Borgvn 23	Саннес	
3	Петтерсен	Кари	Storgt 20	Ставангер	

Пример изменения типа данных

Теперь мы хотим изменить тип данных столбца с именем «DateOfBirth» в таблице «Persons».

Мы используем следующий оператор SQL:

ИЗМЕНИТЬ ТАБЛИЦУ Persons
ALTER COLUMN DateOfBirth year

Обратите внимание, что столбец "DateOfBirth" теперь имеет тип year и будет содержать год в двухзначном или четырехзначном формате.

Удалить столбец Пример

Далее, мы хотим удалить столбец с именем "DateOfBirth" в таблице

"Persons". Мы используем следующий оператор SQL:

ИЗМЕНИТЬ ТАБЛИЦУ
Persons УДАЛИТЬ
СТОЛБЕЦ DateOfBirth

Таблица "Persons" теперь будет выглядеть так:

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер

АВТОУВЕЛИЧЕНИЕ поля

- Автоувеличение позволяет генерировать уникальный номер при вставке новой записи в таблицу.
- Очень часто нам нужно, чтобы значение поля первичного ключа создавалось автоматически каждый раз при вставке новой записи.

Мы хотим создать поле с автоувеличением в таблице.

Синтаксис для MySQL

Следующая SQL-инструкция определяет столбец «P_Id» как поле первичного ключа с автоинкрементом в таблице «Persons»:

```
CREATE TABLE Persons
(
P_Id int NOT NULL AUTO_INCREMENT,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Адрес varchar(255),
Город varchar(255),
Первичный ключ (P_Id)
)
```

MySQL использует ключевое слово AUTO_INCREMENT для реализации функции автоувеличения.

По умолчанию начальное значение для AUTO_INCREMENT равно 1, и оно будет увеличиваться на 1 для каждой новой записи.

Чтобы последовательность AUTO_INCREMENT начиналась с другого значения, используйте

следующую инструкцию SQL: ALTER TABLE Persons AUTO_INCREMENT=100

Чтобы вставить новую запись в таблицу "Persons", нам не нужно указывать значение для столбца "P_Id" (уникальное значение будет добавлено автоматически):

```
INSERT INTO Persons
(FirstName,LastName) VALUES
('Lars','Monsen')
```

Вышеприведенная SQL-команда вставит новую запись в таблицу «Persons». Столбцу «P_Id» будет присвоено уникальное значение. Столбец «FirstName» будет установлен в значение «Lars», а столбец «LastName» — в значение «Monsen».

Синтаксис для SQL Server

Следующая SQL-инструкция определяет столбец «P_Id» как поле первичного ключа с автоинкрементом в таблице «Persons»:

```
CREATE TABLE Persons
(
```

P_Id int PRIMARY KEY IDENTITY,

```
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Адрес varchar(255),
Город varchar(255)
)
```

MS SQL Server использует ключевое слово IDENTITY для реализации функции автоувеличения.

По умолчанию начальное значение для IDENTITY равно 1, и оно будет увеличиваться на 1 для каждой новой записи.

Чтобы указать, что столбец "P_Id" должен начинаться со значения 10 и увеличиваться на 5, измените идентификатор на IDENTITY(10,5).

Чтобы вставить новую запись в таблицу "Persons", нам не нужно указывать значение для столбца "P_Id" (уникальное значение будет добавлено автоматически):

```
INSERT INTO Persons
(FirstName,LastName) VALUES
('Lars','Monsen')
```

Вышеприведенная SQL-команда вставит новую запись в таблицу «Persons». Столбцу «P_Id» будет присвоено уникальное значение. Столбец «FirstName» будет установлен в значение «Lars», а столбец «LastName» — в значение «Monsen».

Синтаксис для Access

Следующая SQL-инструкция определяет столбец «P_Id» как поле первичного ключа с автоинкрементом в таблице «Persons»:

```
CREATE TABLE Persons
(
P_Id PRIMARY KEY AUTOINCREMENT,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Адрес varchar(255),
Город varchar(255)
)
```

MS Access использует ключевое слово AUTOINCREMENT для реализации функции автоувеличения.

По умолчанию начальное значение для AUTOINCREMENT равно 1, и оно будет увеличиваться на 1 для каждой новой записи.

Чтобы указать, что столбец "P_Id" должен начинаться со значения 10 и увеличиваться на 5, измените автоувеличение на AUTOINCREMENT(10,5).

Чтобы вставить новую запись в таблицу "Persons", нам не нужно указывать значение для столбца "P_Id" (уникальное значение будет добавлено автоматически):

```
INSERT INTO Persons
(FirstName,LastName) VALUES
('Lars','Monsen')
```

Вышеприведенная SQL-команда вставит новую запись в таблицу «Persons». Столбцу «P_Id» будет присвоено уникальное значение. Столбец «FirstName» будет установлен в значение «Lars», а столбец «LastName» — в значение «Monsen».

Синтаксис для Oracle

В Oracle код немного сложнее.

Вам придется создать поле с автоувеличением с помощью объекта sequence (этот объект генерирует числовую последовательность).

Используйте следующий синтаксис CREATE SEQUENCE:

```
CREATE SEQUENCE seq_person
MINVALUE 1
START WITH 1
INCREMENT BY 1
CACHE 10
```

Приведенный выше код создает объект последовательности с именем seq_person, который начинается с 1 и будет увеличиваться на 1. Он также будет кэшировать до 10 значений для повышения производительности. Параметр cache указывает, сколько значений последовательности будет храниться в памяти для более быстрого доступа.

Чтобы вставить новую запись в таблицу "Persons", нам нужно будет использовать функцию nextval (эта функция извлекает следующее значение из последовательности seq_person):

```
INSERT INTO Persons
(P_Id,FirstName,LastName) VALUES
(seq_person.nextval,'Lars','Monsen')
```

Вышеприведенная SQL-команда вставит новую запись в таблицу «Persons». Столбцу «P_Id» будет присвоено следующее число из последовательности seq_person. Столбец «FirstName» будет установлен в значение «Lars», а столбец «LastName» — в значение «Monsen».

Оператор SQL CREATE VIEW

- В SQL представление - это виртуальная таблица, основанная на результирующем наборе инструкции SQL.
- Представление содержит строки и столбцы, как и реальная таблица. Поля в представлении являются полями из одной или нескольких реальных таблиц в базе данных.
- Вы можете добавлять SQL-функции, WHERE и JOIN операторы к представлению и представлять данные так, как если бы они поступали из одной таблицы.
- Синтаксис SQL CREATE VIEW:

```
CREATE VIEW view_name AS
SELECT column_name(s)
```

ИЗ таблицы_название
ГДЕ условие

Примечание:Представление всегда отображает актуальные данные! СУБД воссоздает данные, используя инструкцию SQL представления, каждый раз, когда пользователь запрашивает представление.

Примеры SQL CREATE VIEW

Если у вас есть база данных Northwind, вы можете увидеть, что в ней по умолчанию установлено несколько представлений.

Представление "Current Product List" перечисляет все активные продукты (продукты, которые не сняты с производства) из таблицы "Products". Представление создается с помощью следующего SQL-кода:

```
CREATE VIEW [Current Product List] AS
SELECT ProductID,ProductName
ИЗ Products
WHERE Discontinued=No
```

Мы можем запросить представление выше следующим образом:

```
SELECT * FROM [Current Product List]
```

Другое представление в базе данных Northwind выбирает каждый продукт в таблице "Products" с ценой за единицу выше средней цены за единицу:

```
CREATE VIEW [Products Above Average Price] AS
SELECT ProductName,UnitPrice
ИЗ Products
WHERE UnitPrice>(SELECT AVG(UnitPrice) FROM Products)
```

Мы можем запросить представление выше следующим

образом:

```
SELECT * FROM [Products Above Average Price]
```

Другое представление в базе данных Northwind вычисляет общую сумму продаж для каждой категории в 1997 году. Обратите внимание, что это представление выбирает свои данные из другого представления под названием "Product Sales for 1997":

```
CREATE VIEW [Category Sales For 1997] AS
SELECT DISTINCT CategoryName,Sum(ProductSales) AS CategorySales
FROM [Product Sales for 1997]
GROUP BY CategoryName
```

Мы можем запросить представление выше следующим образом:

SELECT * FROM [Category Sales For 1997]

Мы также можем добавить условие к запросу. Теперь мы хотим видеть общую сумму продаж только для категории "Beverages":

```
SELECT * FROM [Category Sales For 1997]
WHERE CategoryName='Beverages'
```

Обновление представления SQL

Вы можете обновить представление, используя следующий синтаксис:

Синтаксис SQL CREATE OR REPLACE VIEW:

```
CREATE OR REPLACE VIEW view_name AS
SELECT column_name(s)
ИЗ
таблицы_названи
е ГДЕ условие
```

Теперь мы хотим добавить столбец "Category" в представление "Current Product List". Мы обновим представление с помощью следующего SQL-кода:

```
CREATE VIEW [Current Product List] AS
SELECT ProductID,ProductName,Category
FROM Products
WHERE Discontinued=No
```

Удаление представления SQL

Вы можете удалить представление с помощью

команды DROP VIEW. Синтаксис SQL DROP VIEW:

```
DROP VIEW view_name
```

Даты SQL

Самая сложная часть при работе с датами - это убедиться, что формат даты, которую вы пытаетесь вставить, соответствует формату столбца даты в базе данных.

Пока ваши данные содержат только часть даты, ваши запросы будут работать должным образом. Однако, если задействована часть времени, все становится сложнее.

Прежде чем говорить о сложностях запроса дат, мы рассмотрим наиболее важные встроенные функции для работы с датами.

Функции работы с датами в MySQL

В следующей таблице перечислены наиболее важные встроенные функции для работы с датами в MySQL:

Функция	Описание
NOW()	Возвращает текущие дату и время
CURDATE()	Возвращает текущую дату
CURTIME()	Возвращает текущее время
ДАТА()	Извлекает часть даты из выражения даты или даты/времени
ИЗВЛЕЧЬ()	Возвращает отдельную часть даты/времени
DATE_ADD()	Добавляет заданный интервал времени к дате
DATE_SUB()	Вычитает заданный интервал времени из даты
DATEDIFF()	Возвращает количество дней между двумя датами
DATE_FORMAT()	Отображает данные даты/времени в различных форматах

Функции работы с датами в SQL Server

В следующей таблице перечислены наиболее важные встроенные функции для работы с датами в SQL Server:

Функция	Описание
GETDATE()	Возвращает текущие дату и время
DATEPART()	Возвращает отдельную часть даты/времени
DATEADD()	Добавляет или вычитает заданный интервал времени из даты
DATEDIFF()	Возвращает время между двумя датами
CONVERT()	Отображает данные даты/времени в различных форматах

Типы данных даты в SQL

MySQLпредлагает следующие типы данных для хранения даты или значения даты/времени в базе данных:

- ДАТА - формат ГГГГ-ММ-ДД
- DATETIME - формат: ГГГГ-ММ-ДД ЧЧ:ММ:СС
- TIMESTAMP - формат: ГГГГ-ММ-ДД ЧЧ:ММ:СС
- YEAR - формат ГГГГ или ГГ

SQL Serverпредлагает следующие типы данных для хранения даты или значения даты/времени в базе данных:

- ДАТА - формат ГГГГ-ММ-ДД
- DATETIME - формат: ГГГГ-ММ-ДД ЧЧ:ММ:СС

- SMALLDATETIME - формат: ГГГГ-ММ-ДД ЧЧ:ММ:СС
- TIMESTAMP - формат: уникальное число

Примечание:Типы данных для столбца выбираются при создании новой таблицы в

базе данных! Обзор всех доступных типов данных.

Работа с датами в SQL

Вы можете легко сравнить две даты, если в них не указано время!

Предположим, что у нас есть следующая таблица «Заказы»:

Код заказа	Название продукта	ДатаЗаказа
1	Гейтост	2008-11-11
2	Камамбер Пьеро	2008-11-09
3	Моцарелла ди Джованни	2008-11-11
4	Маскарпоне Фабиоли	2008-10-29

Теперь мы хотим выбрать записи из таблицы выше с OrderDate равной "2008-11-11".

Мы используем следующий оператор SELECT:

SELECT * FROM Orders WHERE OrderDate='2008-11-11'

Результирующий набор будет выглядеть так:

Код заказа	Название продукта	ДатаЗаказа
1	Гейтост	2008-11-11
3	Моцарелла ди Джованни	2008-11-11

Теперь предположим, что таблица "Orders" выглядит следующим образом (обратите внимание на компонент времени в столбце "OrderDate"):

Код заказа	Название продукта	ДатаЗаказа
1	Гейтост	2008-11-11 13:23:44
2	Камамбер Пьеро	2008-11-09 15:45:21
3	Моцарелла ди Джованни	2008-11-11 11:12:01
4	Маскарпоне Фабиоли	2008-10-29 14:56:59

Если мы используем тот же оператор SELECT, что и выше:

SELECT * FROM Orders WHERE OrderDate='2008-11-11'

мы не получим никакого результата! Это связано с тем, что запрос ищет только даты без временной составляющей.

Совет:Если вы хотите, чтобы ваши запросы были простыми и удобными в обслуживании, не допускайте наличия временных компонентов в ваших датах!

Определение и использование

NOW()возвращает текущую дату и время.

Синтаксис
NOW()

Пример

Следующий оператор SELECT:

SELECT NOW(),CURDATE(),CURTIME()

приведет к примерно следующему результату:

NOW()	CURDATE()	CURTIME()
2008-11-11 12:45:34	2008-11-11	12:45:34

Пример

Следующий SQL-запрос создает таблицу "Orders" со столбцом типа datetime (OrderDate):

CREATE TABLE
Заказы (
OrderId int NOT NULL,
ProductName varchar(50) NOT NULL,
OrderDate datetime NOT NULL DEFAULT
NOW(), PRIMARY KEY (OrderId)
)

Обратите внимание, что для столбца OrderDate в качестве значения по умолчанию указано NOW(). В результате, когда вы вставляете строку в таблицу, текущая дата и время автоматически вставляются в столбец.

Теперь мы хотим вставить запись в таблицу «Orders»:

INSERT INTO Orders (ProductName) VALUES ('Jarlsberg

Cheese') Таблица "Orders" теперь будет выглядеть примерно

так:

Код заказа	Название продукта	ДатаЗаказа

1	Сыр Ярлсберг	2008-11-11 13:23:44.657

Определение и использование

CURDATE()возвращает текущую

дату. Синтаксис
CURDATE()

Пример

Следующий оператор SELECT:

SELECT NOW(),CURDATE(),CURTIME()

приведет к примерно следующему результату:

NOW()	CURDATE()	CURTIME()
2008-11-11 12:45:34	2008-11-11	12:45:34

Пример

Следующий SQL-запрос создает таблицу "Orders" со столбцом типа datetime (OrderDate):

CREATE TABLE
Заказы (
OrderId int NOT NULL,
ProductName varchar(50) NOT NULL,
OrderDate datetime NOT NULL DEFAULT CURDATE(),
PRIMARY KEY (OrderId)
)

Обратите внимание, что столбец OrderDate указывает CURDATE() в качестве значения по умолчанию. В результате, когда вы вставляете строку в таблицу, текущая дата автоматически вставляется в столбец.

Теперь мы хотим вставить запись в таблицу «Orders»:

INSERT INTO Orders (ProductName) VALUES ('Jarlsberg

Cheese') Таблица "Orders" теперь будет выглядеть примерно

так:

Код заказа	Название продукта	ДатаЗаказа
1	Сыр Ярлсберг	2008-11-11

Определение и использование

CURTIME()возвращает текущее время.

Синтаксис
CURTIME()

Пример

Следующий оператор SELECT:

SELECT NOW(),CURDATE(),CURTIME()

приведет к примерно следующему результату:

NOW()	CURDATE()	CURTIME()
2008-11-11 12:45:34	2008-11-11	12:45:34

Определение и использование

Функция DATE() извлекает часть даты из выражения даты или даты/времени.

Синтаксис
DATE(date)

Где date - допустимое выражение даты.

Пример

Предположим, что у нас есть следующая таблица «Заказы»:

Код заказа	Название продукта	ДатаЗаказа
1	Сыр Ярлсберг	2008-11-11 13:23:44.657

Следующий оператор SELECT:

SELECT ProductName, DATE(OrderDate) AS
OrderDate FROM Orders
WHERE OrderId=1

приведет к

следующему

результату:

Название продукта	ДатаЗаказа
Сыр Ярлсберг	2008-11-11

Определение и использование

Функция**EXTRACT()**используется для возврата отдельной части даты/времени, такой как год, месяц, день, час, минута и т. д.

Синтаксис
EXTRACT(unit FROM date)

Где date - допустимое выражение даты, а unit может быть одним из следующих:

Значение единицы измерения
МИКРОСЕКУНДА
ВТОРОЙ
МИНУТА
ЧАС
ДЕНЬ
НЕДЕЛЯ
МЕСЯЦ
КВАРТАЛ
ГОД
МИКРОСЕКУНДА$_2$
МИНУТА_МИКРОСЕКУНДА
МИНУТЫ_СЕКУНДЫ
ЧАС_МИКРОСЕКУНДА
ЧАС_СЕКУНДА

ЧАС_МИНУТА
DAY_MICROSECOND
ДЕНЬ_СЕКУНДА
DAY_MINUTE
ЧАС_ДНЯ
ГОД_МЕСЯЦ

Пример

Предположим, что у нас есть следующая таблица «Заказы»:

Код заказа	Название продукта	ДатаЗаказа
1	Сыр Ярлсберг	2008-11-11 13:23:44.657

Следующий оператор SELECT:

```
SELECT EXTRACT(YEAR FROM OrderDate) AS
OrderYear, EXTRACT(MONTH FROM OrderDate) AS
OrderMonth, EXTRACT(DAY FROM OrderDate) AS
OrderDay,
ИЗ Orders ГДЕ
OrderId=1
```

приведет к следующему:

Год заказа	МесяцЗаказа	ДеньЗаказа
2008	11	11

Определение и использование

Функция**DATE_ADD()**добавляет указанный интервал времени к дате.

Синтаксис
DATE_ADD(date,INTERVAL expr type)

Где date – допустимое выражение даты, а expr – число интервалов, которое

необходимо добавить. Тип может быть одним из следующих:

Значение типа
МИКРОСЕКУНДА
ВТОРОЙ
МИНУТА
ЧАС
ДЕНЬ
НЕДЕЛЯ
МЕСЯЦ
КВАРТАЛ
ГОД
МИКРОСЕКУНДА$_2$
МИНУТА_МИКРОСЕКУНДА
МИНУТЫ_СЕКУНДЫ
ЧАС_МИКРОСЕКУНДА
ЧАС_СЕКУНДА
ЧАС_МИНУТА
DAY_MICROSECOND
ДЕНЬ_СЕКУНДА
DAY_MINUTE
ЧАС_ДНЯ
ГОД_МЕСЯЦ

Пример

Предположим, что у нас есть следующая таблица «Заказы»:

Код заказа	Название продукта	ДатаЗаказа
1	Сыр Ярлсберг	2008-11-11 13:23:44.657

Теперь мы хотим добавить 45 дней к «OrderDate», чтобы найти дату

оплаты. Мы используем следующий оператор SELECT:

```
SELECT OrderId,DATE_ADD(OrderDate,INTERVAL 45 DAY) AS OrderPayDate
FROM Orders
```

Результат:

Код заказа	ДатаОплатыЗаказа
1	2008-12-26 13:23:44.657

Определение и использование

Функция**DATE_SUB()**вычитает указанный интервал времени из даты.

Синтаксис
```
DATE_SUB(date,INTERVAL expr type)
```

Где date – допустимое выражение даты, а expr – число интервалов, которое необходимо

вычесть. Тип может быть одним из следующих:

Значение типа
МИКРОСЕКУНДА
ВТОРОЙ
МИНУТА
ЧАС
ДЕНЬ
НЕДЕЛЯ
МЕСЯЦ

КВАРТАЛ
ГОД
МИКРОСЕКУНДА[2]
МИНУТА_МИКРОСЕКУНДА
МИНУТЫ_СЕКУНДЫ
ЧАС_МИКРОСЕКУНДА
ЧАС_СЕКУНДА
ЧАС_МИНУТА
DAY_MICROSECOND
ДЕНЬ_СЕКУНДА
DAY_MINUTE
ЧАС_ДНЯ
ГОД_МЕСЯЦ

Пример

Предположим, что у нас есть следующая таблица «Заказы»:

Код заказа	Название продукта	ДатаЗаказа
1	Сыр Ярлсберг	2008-11-11 13:23:44.657

Теперь мы хотим вычесть 5 дней из даты "OrderDate".

Мы используем следующий оператор SELECT:

```
SELECT OrderId,DATE_SUB(OrderDate,INTERVAL 5 DAY) AS SubtractDate
FROM Orders
```

Результат:

Код заказа	ВычитаниеДаты
1	2008-11-06 13:23:44.657

Определение и использование

Функция**DATEDIFF()**возвращает разницу между двумя датами.

Синтаксис
DATEDIFF(date1,date2)

Где date1 и date2 являются допустимыми выражениями даты или даты/времени.

Примечание:В расчете используются только части даты значений.

Пример

Следующий оператор SELECT:

SELECT DATEDIFF('2008-11-30','2008-11-29') AS DiffDate

приведет к следующему:

DiffDate
1

Пример

Следующий оператор SELECT:

SELECT DATEDIFF('2008-11-29','2008-11-30') AS DiffDate

приведет к следующему:

DiffDate
-1

Определение и использование

Функция**DATE_FORMAT()**используется для отображения данных даты/времени в различных форматах.

Синтаксис
DATE_FORMAT(date,format)

Где date - допустимая дата, а format задает формат вывода для даты/времени.

Могут использоваться следующие форматы:

Формат	Описание
%a	Сокращённое название дня недели
%b	Сокращенное название месяца
%c	Месяц, числовой
%D	День месяца с английским суффиксом
%d	Число месяца, числовое (00-31)
%e	Число месяца (0-31)
%f	Микросекунды
%H	Час (00-23)
%h	Час (01-12)
%I	Час (01-12)
%i	Минуты, числовое значение (00-59)
%j	День года (001-366)
%k	Час (0-23)
%l	Час (1-12)
%M	Название месяца
%m	Месяц, числовой (00-12)
%p	AM или PM
%r	Время, 12-часовой (чч:мм:сс AM или PM)

%S	Секунды (00-59)
%s	Секунды (00-59)
%T	Время, 24-часовой (чч:мм:сс)
%U	Номер недели (00-53), где воскресенье – первый день недели
%u	Номер недели (00-53), где понедельник — первый день недели
%V	Номер недели (01-53), где воскресенье – первый день недели, используется с %X
%v	Номер недели (01-53), где понедельник является первым днем недели, используется с %x
%W	Название дня недели
%w	День недели (0=воскресенье, 6=суббота)
%X	Номер недели в году, где воскресенье является первым днем недели, четыре цифры, используется с %V
%x	Номер недели в году, где понедельник — первый день недели, четыре цифры, используется с %v
%Y	Год (четыре цифры)
%y	Год, две цифры

Пример

В следующем скрипте используется функция DATE_FORMAT() для отображения различных форматов. Мы будем использовать функцию NOW() для получения текущей даты/времени:

```
DATE_FORMAT(NOW(),'%b %d %Y %h:%i %p')
DATE_FORMAT(NOW(),'%m-%d-%Y')
DATE_FORMAT(NOW(),'%d %b %y')
DATE_FORMAT(NOW(),'%d %b %Y %T:%f')
```

Результат будет выглядеть примерно так:

```
04 ноя 2008 23:45
11-04-2008
04 ноя 08
04 нояб. 2008 г. 11:45:34:243
```

Определение и использование

Функция**GETDATE()**возвращает текущую дату и время из SQL Server.

Синтаксис
GETDATE()

Пример

Следующий оператор SELECT: SELECT

GETDATE() AS CurrentDateTime

приведет к результату, подобному этому:

CurrentDateTime
2008-11-11 12:45:34.243

Примечание:Временная часть выше отображается вплоть до миллисекунд.

Пример

Следующий SQL-запрос создает таблицу "Orders" со столбцом типа datetime (OrderDate):

```
CREATE TABLE
Заказы (
OrderId int NOT NULL PRIMARY KEY,
ProductName varchar(50) NOT NULL,
OrderDate datetime NOT NULL DEFAULT GETDATE()
)
```

Обратите внимание, что для столбца OrderDate в качестве значения по умолчанию указана функция GETDATE(). В результате, когда вы вставляете строку в таблицу, текущая дата и время автоматически вставляются в столбец.

Теперь мы хотим вставить запись в таблицу «Orders»:

INSERT INTO Orders (ProductName) VALUES ('Jarlsberg

Cheese') Таблица "Orders" теперь будет выглядеть примерно

так:

Код заказа	Название продукта	ДатаЗаказа

1	Сыр Ярлсберг	2008-11-11 13:23:44.657

Определение и использование

Функция**DATEPART()**используется для возврата отдельной части даты/времени, такой как год, месяц, день, час, минута и т. д.

Синтаксис
DATEPART(datepart,date)

Где date - допустимое выражение даты, а datepart может быть одним из следующих:

часть даты	**Сокращение**
год	гг, гггг
квартал	qq, q
месяц	мм, м
деньГода	dy, y
день	дд, д
неделя	нк, вв
день недели	dw, w
час	чч
минута	mi, n
второй	сс, с
миллисекунда	мс
микросекунда	mcs
наносекунда	нс

Пример

Предположим, что у нас есть следующая таблица «Заказы»:

Код заказа	Название продукта	ДатаЗаказа
1	Сыр Ярлсберг	2008-11-11 13:23:44.657

Следующий оператор SELECT:

```
SELECT DATEPART(yyyy,OrderDate) AS OrderYear,
DATEPART(mm,OrderDate) AS OrderMonth,
DATEPART(dd,OrderDate) AS OrderDay,
ИЗ Orders ГДЕ
OrderId=1
```

приведет к следующему:

Год заказа	МесяцЗаказа	ДеньЗаказа
2008	11	11

Определение и использование

Функция**DATEADD()**добавляет или вычитает указанный интервал времени из даты.

Синтаксис
DATEADD(datepart,number,date)

Где date - допустимое выражение даты, а number - количество интервалов, которое вы хотите добавить. Число может быть положительным для дат в будущем или отрицательным для дат в прошлом.

datepart может быть одним из следующих:

часть даты	Сокращение
год	гг, гггг
квартал	qq, q
месяц	мм, м
деньГода	dy, y
день	дд, д

неделя	нк, вв
день недели	dw, w
час	чч
минута	mi, n
второй	сс, с
миллисекунда	мс
микросекунда	mcs
наносекунда	нс

Пример

Предположим, что у нас есть следующая таблица «Заказы»:

Код заказа	Название продукта	ДатаЗаказа
1	Сыр Ярлсберг	2008-11-11 13:23:44.657

Теперь мы хотим добавить 45 дней к «OrderDate», чтобы найти дату

оплаты. Мы используем следующий оператор SELECT:

```
SELECT OrderId,DATEADD(day,45,OrderDate) AS OrderPayDate
FROM Orders
```

Результат:

Код заказа	ДатаОплатыЗаказа
1	2008-12-26 13:23:44.657

Определение и использование

Функция**DATEDIFF()**возвращает разницу между двумя датами.

Синтаксис

DATEDIFF(datepart,startdate,enddate)

Где startdate и enddate являются допустимыми выражениями даты, а datepart может быть одним из следующих:

часть даты	Сокращение
год	гг, гггг
квартал	qq, q
месяц	мм, м
деньГода	dy, y
день	дд, д
неделя	нк, вв
день недели	dw, w
час	чч
минута	mi, n
второй	cc, c
миллисекунда	мс
микросекунда	mcs
наносекунда	нс

Пример

Теперь мы хотим получить количество дней между

двумя датами. Мы используем следующий оператор

SELECT:

SELECT DATEDIFF(day,'2008-06-05','2008-08-05') AS DiffDate

Результат:

DiffDate
61

Пример

Теперь мы хотим получить количество дней между двумя датами (обратите внимание, что вторая дата "раньше" первой, и результатом будет отрицательное число).

Мы используем следующий оператор SELECT:

SELECT DATEDIFF(day,'2008-08-05','2008-06-05') AS DiffDate

Результат:

DiffDate
-61

Определение и использование

Функция**CONVERT()**- это общая функция для преобразования данных в новый тип

данных. Функция CONVERT() может использоваться для отображения данных

даты/времени в различных форматах.

Синтаксис
CONVERT(data_type(length),data_to_be_converted,style)

Где data_type(length) указывает целевой тип данных (с необязательной длиной), data_to_be_converted содержит преобразуемое значение, а style указывает формат вывода для даты/времени.

Можно использовать следующие стили:

Идентификатор стиля	Формат стиля
100 или 0	дд мм гггг чч:ммAM (или PM)
101	дд/мм/гг
102	yy.mm.dd
103	дд/мм/гг

104	dd.mm.yy
105	дд-мм-гг
106	дд мес гг
107	пн дд, гг
108	чч:мм:сс
109 или 9	дд мм гггг чч:мм:сс:ммммAM (или PM)
110	дд-мм-гг
111	гг/мм/дд
112	ггммдд
113 или 13	дд мес гггг чч:мм:сс:мсч(24ч)
114	чч:мм:сс:мс(24ч)
120 или 20	гггг-мм-дд чч:мм:сс(24ч)
121 или 21	гггг-мм-дд чч:мм:сс.ммм(24ч)
126	гггг-мм-ддTчч:мм:сс.ммм(без пробелов)
130	дд мес гггг чч:мм:сс:ммммAM
131	дд/мм/гг чч:мм:сс:ммммAM

Пример

Следующий скрипт использует функцию CONVERT() для отображения различных форматов. Мы будем использовать функцию GETDATE() для получения текущей даты/времени:

```
CONVERT(VARCHAR(19),GETDATE())
CONVERT(VARCHAR(10),GETDATE(),110)
CONVERT(VARCHAR(11),GETDATE(),106)
CONVERT(VARCHAR(24),GETDATE(),113)
```

Результат будет выглядеть примерно так:

```
04 ноя 2008 23:45
```

11-04-2008
04 ноя 08
04 нояб. 2008 г. 11:45:34:243

Значения NULL представляют отсутствующие неизвестные данные.По умолчанию столбец таблицы может содержать значения NULL.

Значения NULL в SQL

- Если столбец в таблице является необязательным, мы можем вставить новую запись или обновить существующую запись, не добавляя значение в этот столбец. Это означает, что поле будет сохранено со значением NULL.
- Значения NULL обрабатываются иначе, чем другие значения.
- NULL используется в качестве заполнителя для неизвестных или неприменимых значений.
- **Примечание:**Невозможно сравнивать NULL и 0; они не эквивалентны.

Работа с NULL значениями в SQL

Рассмотрим следующую таблицу "Persons":

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля		Саннес
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари		Ставангер

Предположим, что столбец "Address" в таблице "Persons" является необязательным. Это означает, что если мы вставим запись без значения для столбца "Address", столбец "Address" будет сохранен со значением NULL.

Как мы можем проверить наличие значений NULL?

Невозможно проверить значения NULL с помощью операторов сравнения, таких как

=, < или >. Вместо этого нам придется использовать операторы IS NULL и IS NOT

NULL.

SQL IS NULL

Как выбрать только те записи, у которых в столбце "Address" есть

значения NULL? Нам нужно использовать оператор IS NULL:

```
SELECT LastName,FirstName,Address FROM Persons
WHERE Address IS NULL
```

Результирующий набор будет выглядеть так:

Фамилия	Имя	Адрес
Хансен	Оля	

Петтерсен	Кари	

Совет:Всегда используйте IS NULL для поиска значений NULL.

SQL IS NOT NULL

Как выбрать только те записи, у которых в столбце "Address" нет значений

NULL? Нам нужно использовать оператор IS NOT NULL:

```
SELECT LastName,FirstName,Address FROM Persons
WHERE Address IS NOT NULL
```

Результирующий набор будет выглядеть так:

Фамилия	Имя	Адрес
Свендсон	Туве	Borgvn 23

В следующей главе мы рассмотрим функции ISNULL(), NVL(), IFNULL() и COALESCE().

Функции SQL ISNULL(), NVL(), IFNULL() и COALESCE()

Рассмотрим следующую таблицу "Products":

P_Id	Название продукта	Цена за единицу	Количество НаСкладе	КоличествоЗаказанныхЕдиниц
1	Ярлсберг	10.45	16	15
2	Маскарпоне	32.56	23	
3	Горгонзола	15.67	9	20

Предположим, что столбец "UnitsOnOrder" является необязательным и может

содержать значения NULL. У нас есть следующий оператор SELECT:

```
SELECT ProductName,UnitPrice*(UnitsInStock+UnitsOnOrder)
FROM Products
```

В приведенном выше примере, если какое-либо из значений "UnitsOnOrder" равно NULL,

результатом будет NULL. Функция ISNULL() от Microsoft используется для указания того,

как мы хотим обрабатывать значения NULL.

Функции NVL(), IFNULL() и COALESCE() также могут быть использованы для

достижения того же результата. В этом случае мы хотим, чтобы значения NULL были

равны нулю.

Ниже, если "UnitsOnOrder" равно NULL, это не повредит вычислению, потому что ISNULL() возвращает ноль, если значение равно NULL:

SQL Server / MS Access

```
SELECT ProductName,UnitPrice*(UnitsInStock+ISNULL(UnitsOnOrder,0))
FROM Products
```

Oracle

В Oracle нет функции ISNULL(). Однако мы можем использовать функцию NVL() для достижения того же результата:

```
SELECT ProductName,UnitPrice*(UnitsInStock+NVL(UnitsOnOrder,0))
FROM Products
```

MySQL

В MySQL есть функция ISNULL(). Однако она работает немного иначе, чем функция ISNULL() от Microsoft.

В MySQL мы можем использовать функцию IFNULL() следующим образом:

```
SELECT ProductName,UnitPrice*(UnitsInStock+IFNULL(UnitsOnOrder,0))
ИЗ Products
```

или мы можем использовать функцию COALESCE() следующим образом:

```
SELECT ProductName,UnitPrice*(UnitsInStock+COALESCE(UnitsOnOrder,0))
FROM Products
```

Типы данных Microsoft Access

Типы данных и диапазоны для Microsoft Access, MySQL и SQL Server.

Тип данных	Описание	Хранилище
Текст	Используйте для текста или комбинаций текста и чисел.Максимум 255 символов.	
Записка	Тип данных "Memo" используется для больших объемов текста.Хранит до 65 536 символов.**Примечание:Вы не можете сортировать поле типа "Memo".**Однако, по ним можно выполнять поиск.	
Байт	Допускает целые числа от 0 до 255	1 байт
Целое число	Допускает целые числа от -32 768 до 32 767	2 байта
Длинный	Допускает целые числа от -2 147 483 648 до 2 147 483 647	4 байта
Одиночный	Одноточечные числа с плавающей запятой одинарной точности. Обработает большинство десятичных дробей	4 байта
Двойное	Число с плавающей запятой двойной точности. Обработает большинство десятичных дробей	8 байт

Валюта	Используется для валюты.Вмещает до 15 цифр целых долларов и 4 десятичных знака.**Совет:**Вы можете выбрать, валюту какой страны использовать.	8 байт
Автонумерация	Поля «Автонумерация» автоматически присваивают каждой записи свой номер, обычно начиная с 1	4 байта
Дата/Время	Использовать для дат и времени	8 байт
Да/Нет	Логическое поле может отображаться как Да/Нет, Истина/Ложь или Вкл/Выкл.В коде используйте константы True и False (эквивалентны -1 и 0).**Примечание:**значения Null не допускаются в полях типа Да/Нет	1 бит
OLE-объект	Может хранить изображения, аудио, видео или другие BLOB-объекты (большие двоичные объекты)	до 1 ГБ
Гиперссылка	Содержат ссылки на другие файлы, включая веб-страницы	
Мастер подстановки	Позволяет ввести список вариантов, которые затем можно выбрать из выпадающего списка	4 байта

Типы данных MySQL

В MySQL есть три основных типа: текстовый, числовой и типы Дата/Время.

Типы текста:

Тип данных	Описание
CHAR(размер)	Хранит строку фиксированной длины (может содержать буквы, цифры и специальные символы).Фиксированный размер указывается в скобках.Может хранить до 255 символов.
VARCHAR(размер)	Хранит строку переменной длины (может содержать буквы, цифры и специальные символы).Максимальный размер указывается в скобках.Может хранить до 255 символов.**Примечание:**Если вы укажете значение больше 255, оно будет преобразовано в тип TEXT.
TINYTEXT	Содержит строку максимальной длиной 255 символов
ТЕКСТ	Содержит строку максимальной длиной 65 535 символов
BLOB	Для больших двоичных объектов (BLOB). Вмещает до 65 535 байт данных
MEDIUMTEXT	Содержит строку максимальной длиной 16 777 215 символов
MEDIUMBLOB	Для больших двоичных объектов (BLOB). Вмещает до 16 777 215 байт данных
ДЛИННЫЙ ТЕКСТ	Содержит строку максимальной длиной 4 294 967 295 символов
LONGBLOB	Для больших двоичных объектов (BLOB). Вмещает до 4 294 967 295 байт данных

ПЕРЕЧИСЛЕНИЕ(x,y,z,и т.д.)	Позволяет ввести список возможных значений.Вы можете перечислить до 65535 значений в списке ENUM.Если вставляется значение, которого нет в списке, будет вставлено пустое значение. **Примечание:**Значения сортируются в том порядке, в котором вы их вводите. Вы вводите возможные значения в следующем формате: ENUM('X','Y','Z')
УСТАНОВИТЬ	Аналогично ENUM, за исключением того, что SET может содержать до 64 элементов списка и может хранить более одного варианта выбора

Числовые типы:

Тип данных	Описание
TINYINT(размер)	-128 до 127 нормально.0 до 255 БЕЗ ЗНАКА*.Максимальное количество цифр может быть
	указанных в скобках
SMALLINT(размер)	-32768 до 32767 обычный.От 0 до 65535 БЕЗ ЗНАКА*. Максимальное количество цифр может быть указано в скобках
MEDIUMINT(размер)	-8388608 до 8388607 обычный.От 0 до 16777215 БЕЗ ЗНАКА*. Максимальное количество цифр может быть указано в скобках
INT(размер)	-2147483648 до 2147483647 обычный.От 0 до 4294967295 БЕЗ ЗНАКА*. Максимальное количество цифр может быть указано в скобках
BIGINT(размер)	-9223372036854775808 до 9223372036854775807 нормальное.0 до 18446744073709551615 БЕЗ ЗНАКА*. Максимальное количество цифр может быть указано в скобках
FLOAT(размер,d)	Небольшое число с плавающей десятичной точкой. Максимальное количество цифр может быть указано в параметре размера. Максимальное количество цифр справа от десятичной точки указывается в параметре d
DOUBLE(размер,d)	Большое число с плавающей десятичной точкой. Максимальное количество цифр может быть указано в параметре размера. Максимальное количество цифр справа от десятичной точки указывается в параметре d
DECIMAL(размер, d)	Значение типа DOUBLE, хранящееся в виде строки, что позволяет использовать фиксированную десятичную точку. Максимальное количество цифр может быть указано в параметре размера. Максимальное количество цифр справа от десятичной точки указывается в параметре d

*Целочисленные типы имеют дополнительный параметр UNSIGNED (без знака). Обычно целое число принимает значения от отрицательного до положительного. Добавление атрибута UNSIGNED сдвигает этот диапазон так, что он начинается с нуля, а не с отрицательного числа.

Типы данных даты:

Тип данных	Описание

ДАТА()	Дата.Формат: ГГГГ-ММ-ДД **Примечание:**Поддерживаемый диапазон — от '1000-01-01' до '9999-12-31'
DATETIME()	*Комбинация даты и времени. Формат: ГГГГ-ММ-ДД ЧЧ:ММ:СС **Примечание:**Поддерживаемый диапазон — от '1000-01-01 00:00:00' до '9999-12-31 23:59:59'
TIMESTAMP()	*Временная метка.Значения TIMESTAMP хранятся как количество секунд, прошедших с начала эпохи Unix («1970-01-01 00:00:00» UTC). Формат: ГГГГ-ММ-ДД ЧЧ:ММ:СС **Примечание:**Поддерживаемый диапазон — от '1970-01-01 00:00:01' UTC до '2038-01-09 03:14:07' UTC
TIME()	Время.Формат: ЧЧ:ММ:СС **Примечание:**Поддерживаемый диапазон от '-838:59:59' до '838:59:59'
ГОД()	Год в двухзначном или четырехзначном формате. **Примечание:**Допустимые значения в четырехзначном формате: с 1901 по 2155. Допустимые значения в двухзначном формате: с 70 по 69, что соответствует годам с 1970 по 2069.

*Даже если DATETIME и TIMESTAMP возвращают один и тот же формат, они работают совершенно по-разному. В запросе INSERT или UPDATE TIMESTAMP автоматически устанавливается на текущую дату и время.

TIMESTAMP также принимает различные форматы, такие как YYYYMMDDHHMMSS, YYMMDDHHMMSS, YYYYMMDD или YYMMDD.

Типы данных SQL Server

Строки символов:

Тип данных	Описание	Хранилище
char(n)	Строка символов фиксированной длины. Максимум 8000 символов	n
varchar(n)	Строка символов переменной длины. Максимум 8000 символов	
varchar(макс.)	Строка символов переменной длины. Максимум 1 073 741 824 символа	
текст	Строка символов переменной длины. Максимум 2 ГБ текстовых данных	

Строки Юникода:

Тип данных	Описание	Хранилище
nchar(n)	Unicode-данные фиксированной длины. Максимум 4000 символов	
nvarchar(n)	Данные Unicode переменной длины. Максимум 4000 символов	
nvarchar(max)	Данные Unicode переменной длины. Максимум 536 870 912 символов	
ntext	Данные Unicode переменной длины. Максимум 2 ГБ текстовых данных	

Двоичные типы:

Тип данных	Описание	Хранилище
бит	Допускает 0, 1 или NULL	
двоичный(n)	Двоичные данные фиксированной длины. Максимум 8000 байт	
varbinary(n)	Двоичные данные переменной длины. Максимум 8000 байт	

varbinary(max)	Двоичные данные переменной длины. Максимум 2 ГБ	
изображение	Двоичные данные переменной длины. Максимум 2 ГБ	

Числовые типы:

Тип данных	Описание	Хранилище
целое число размером 1 байт	Допускает целые числа от 0 до 255	1 байт
целое малое число	Допускает целые числа от -32 768 до 32 767	2 байта
int	Допускает целые числа от -2 147 483 648 до 2 147 483 647	4 байта
bigint (целое число большого размера)	Допускает целые числа от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807	8 байт
decimal(p,s)	Числа с фиксированной точностью и масштабом. Допускаются числа от $-10^{38}+1$ до $10^{38}-1$. Параметр p указывает максимальное общее количество цифр, которое может быть сохранено (как слева, так и справа от десятичной точки). p должно быть значением от 1 до 38. Значение по умолчанию — 18. Параметр s указывает максимальное количество цифр, хранящихся справа от десятичной точки. s должно быть значением от 0 до p. Значение по умолчанию — 0	5-17 байт
numeric(p,s)	Числа с фиксированной точностью и масштабом. Допускаются числа от $-10^{38}+1$ до $10^{38}-1$. Параметр p указывает максимальное общее количество цифр, которое может быть сохранено (как слева, так и справа от десятичной точки). p должно быть значением от 1 до 38. Значение по умолчанию — 18. Параметр s указывает максимальное количество цифр, хранящихся справа от десятичной точки. s должно быть значением от 0 до p. Значение по умолчанию — 0	5-17 байт
smallmoney	Денежные данные от -214 748,3648 до 214 748,3647	4 байта
деньги	Денежные данные от -922 337 203 685 477,5808 до 922 337 203 685 477,5807	8 байт

float(n)	Числа с плавающей запятой в диапазоне от -1.79E + 308 до 1.79E + 308. Параметр n указывает, должен ли размер поля составлять 4 или 8 байт.float(24) соответствует полю размером 4 байта, а float(53) – полю размером 8 байт.Значение n по умолчанию равно 53.	4 или 8 байт
вещественный	Числа с плавающей запятой от -3.40E + 38 до 3.40E + 38	4 байта

Типы данных даты:

Тип данных	Описание	Хранилище
Дата и время	С 1 января 1753 года по 31 декабря 9999 года с точностью до 3,33 миллисекунд	8 байт
datetime2	С 1 января 0001 года по 31 декабря 9999 года с точностью до 100 наносекунд	6-8 байт
smalldatetime	С 1 января 1900 года по 6 июня 2079 года с точностью до 1 минуты	4 байта
дата	Хранить только дату.С 1 января 0001 года по 31 декабря 9999 года	3 байта
время	Сохранить время с точностью до 100 наносекунд	3-5 байт
datetimeoffset	То же, что и datetime2, с добавлением смещения часового пояса	8-10 байтов
временна́я метка	Хранит уникальный номер, который обновляется каждый раз при создании или	
	изменено.Значение метки времени основано на внутренних часах и не соответствует реальному времени.Каждая таблица может иметь только одну переменную метки времени.	

Другие типы данных:

Тип данных	Описание
sql_variant	Хранит до 8 000 байт данных различных типов, кроме text, ntext и timestamp
уникальный идентификатор	Хранит глобальный уникальный идентификатор (GUID)
xml	Хранит данные в формате XML. Максимум 2 ГБ
курсор	Хранит ссылку на курсор, используемый для операций с базой данных
таблица	Сохраняет результирующий набор для последующей обработки

ФУНКЦИИ SQL

SQL имеет множество встроенных функций для выполнения вычислений с данными.

Агрегатные функции SQL

Агрегатные функции SQL возвращают одно значение, вычисленное на основе

значений в столбце. Полезные агрегатные функции:

- AVG() - Возвращает среднее значение
- COUNT() - Возвращает количество строк
- FIRST() - Возвращает первое значение
- LAST() - Возвращает последнее значение
- MAX() - Возвращает наибольшее значение
- MIN() - Возвращает наименьшее значение
- SUM() - Возвращает сумму

Скалярные функции SQL

Скалярные функции SQL возвращают одно значение на основе

входного значения. Полезные скалярные функции:

- UCASE() - Преобразует поле в верхний регистр
- LCASE() - Преобразует поле в нижний регистр

- MID() - Извлекает символы из текстового поля
- LEN() - Возвращает длину текстового поля
- ROUND() - Округляет числовое поле до указанного количества десятичных знаков
- NOW() - Возвращает текущую системную дату и время
- FORMAT() - Форматирует способ отображения поля

Функция AVG()

Функция AVG() возвращает среднее значение числового

столбца. Синтаксис SQL AVG()
SELECT AVG(column_name) FROM table_name

Пример SQL AVG()

У нас есть следующая таблица «Заказы»:

O_Id	ДатаЗаказа	ЦенаЗаказа	Клиент
1	2008/11/12	1000	Хансен
2	2008/10/23	1600	Нильсен
3	2008/09/02	700	Хансен
4	2008/09/03	300	Хансен
5	2008/08/30	2000	Йенсен
6	2008/10/04	100	Нильсен

Теперь мы хотим найти среднее значение поля «OrderPrice».

Мы используем следующий оператор SQL:

SELECT AVG(OrderPrice) AS OrderAverage FROM Orders

Результирующий набор будет выглядеть следующим

образом:

OrderAverage
950

Теперь мы хотим найти клиентов, у которых значение OrderPrice выше среднего значения OrderPrice.

Мы используем следующий оператор SQL:

SELECT Customer FROM Orders
WHERE OrderPrice>(SELECT AVG(OrderPrice) FROM Orders)

Результирующий набор будет выглядеть следующим образом:

Клиент
Хансен
Нильсен
Йенсен

Подсчет SQL

Функция COUNT() возвращает количество строк, соответствующих заданному

критерию. Синтаксис SQL COUNT(название_столбца)

Функция COUNT(column_name) возвращает количество значений (значения NULL не учитываются) указанного столбца:

SELECT COUNT(column_name) FROM table_name

Синтаксис SQL COUNT(*)

Функция COUNT(*) возвращает количество записей в таблице:

SELECT COUNT(*) FROM table_name

Синтаксис SQL COUNT(DISTINCT имя_столбца)

Функция COUNT(DISTINCT column_name) возвращает количество уникальных значений указанного столбца:

SELECT COUNT(DISTINCT column_name) FROM table_name

Примечание:COUNT(DISTINCT) работает с ORACLE и Microsoft SQL Server, но не с Microsoft Access.

Пример SQL COUNT(название_столбца)

У нас есть следующая таблица «Заказы»:

O_Id	ДатаЗаказа	ЦенаЗаказа	Клиент
1	2008/11/12	1000	Хансен
2	2008/10/23	1600	Нильсен
3	2008/09/02	700	Хансен
4	2008/09/03	300	Хансен
5	2008/08/30	2000	Йенсен
6	2008/10/04	100	Нильсен

Теперь мы хотим посчитать количество заказов от "Customer

Nilsen". Мы используем следующий оператор SQL:

```
SELECT COUNT(Customer) AS CustomerNilsen FROM Orders
WHERE Customer='Nilsen'
```

Результат приведенного выше оператора SQL будет равен 2, потому что клиент Nilsen сделал всего 2 заказа:

CustomerNilsen
2

Пример SQL COUNT(*)

Если мы опустим предложение WHERE, например так:

```
SELECT COUNT(*) AS NumberOfOrders FROM Orders
```

Результирующий набор будет выглядеть следующим

образом:

NumberOfOrders

6

что является общим количеством строк в таблице.

Пример SQL COUNT(DISTINCT имя_столбца)

Теперь мы хотим посчитать количество уникальных клиентов в

таблице "Orders". Мы используем следующий оператор SQL:

SELECT COUNT(DISTINCT Customer) AS NumberOfCustomers FROM Orders

Результирующий набор будет выглядеть следующим образом:

NumberOfCustomers
3

что является количеством уникальных клиентов (Hansen, Nilsen и Jensen) в таблице "Orders".

Функция FIRST()

Функция FIRST() возвращает первое значение выбранного

столбца. Синтаксис SQL FIRST()
SELECT FIRST(column_name) FROM table_name

Пример SQL FIRST()

У нас есть следующая таблица «Заказы»:

O_Id	ДатаЗаказа	ЦенаЗаказа	Клиент
1	2008/11/12	1000	Хансен
2	2008/10/23	1600	Нильсен
3	2008/09/02	700	Хансен
4	2008/09/03	300	Хансен
5	2008/08/30	2000	Йенсен
6	2008/10/04	100	Нильсен

Теперь мы хотим найти первое значение столбца

«OrderPrice». Мы используем следующий оператор SQL:

SELECT FIRST(OrderPrice) AS FirstOrderPrice FROM

Orders**Совет:**Обходной путь, если функция FIRST() не

поддерживается: SELECT OrderPrice FROM Orders ORDER

BY O_Id LIMIT 1 Результирующий набор будет выглядеть

следующим образом:

FirstOrderPrice
1000

Функция LAST()

Функция LAST() возвращает последнее значение выбранного

столбца. Синтаксис SQL LAST()
SELECT LAST(column_name) FROM table_name

Пример SQL LAST()

У нас есть следующая таблица «Заказы»:

O_Id	ДатаЗаказа	ЦенаЗаказа	Клиент
1	2008/11/12	1000	Хансен
2	2008/10/23	1600	Нильсен
3	2008/09/02	700	Хансен
4	2008/09/03	300	Хансен
5	2008/08/30	2000	Йенсен

6	2008/10/04	100	Нильсен

Теперь мы хотим найти последнее значение в столбце

"OrderPrice". Мы используем следующий оператор SQL:

SELECT LAST(OrderPrice) AS LastOrderPrice FROM Orders

Совет:Обходной путь, если функция LAST() не поддерживается:

SELECT OrderPrice FROM Orders ORDER BY O_Id DESC LIMIT 1

Результирующий набор будет выглядеть следующим образом:

ПоследняяЦенаЗаказа
100

Функция MAX()

Функция MAX() возвращает наибольшее значение выбранного

столбца. Синтаксис SQL MAX()
SELECT MAX(column_name) FROM table_name

Пример SQL MAX()

У нас есть следующая таблица «Заказы»:

O_Id	ДатаЗаказа	ЦенаЗаказа	Клиент
1	2008/11/12	1000	Хансен
2	2008/10/23	1600	Нильсен
3	2008/09/02	700	Хансен
4	2008/09/03	300	Хансен
5	2008/08/30	2000	Йенсен
6	2008/10/04	100	Нильсен

Теперь мы хотим найти наибольшее значение в столбце

"OrderPrice". Мы используем следующий оператор SQL:

SELECT MAX(OrderPrice) AS LargestOrderPrice FROM Orders

Результат будет выглядеть следующим образом:

НаибольшаяЦенаЗаказа
2000

Функция MIN()

Функция MIN() возвращает наименьшее значение выбранного

столбца. Синтаксис SQL MIN()
SELECT MIN(column_name) FROM table_name

Пример SQL MIN()

У нас есть следующая таблица «Заказы»:

O_Id	ДатаЗаказа	ЦенаЗаказа	Клиент
1	2008/11/12	1000	Хансен
2	2008/10/23	1600	Нильсен
3	2008/09/02	700	Хансен
4	2008/09/03	300	Хансен
5	2008/08/30	2000	Йенсен
6	2008/10/04	100	Нильсен

Теперь мы хотим найти наименьшее значение в столбце

"OrderPrice". Мы используем следующий оператор SQL:

SELECT MIN(OrderPrice) AS SmallestOrderPrice FROM Orders

Результат будет выглядеть следующим образом:

НаименьшаяЦенаЗаказа
100

Функция SUM()

Функция SUM() возвращает общую сумму числового

столбца. Синтаксис SQL SUM()
SELECT SUM(column_name) FROM table_name

Пример SQL SUM()

У нас есть следующая таблица «Заказы»:

O_Id	ДатаЗаказа	ЦенаЗаказа	Клиент
1	2008/11/12	1000	Хансен
2	2008/10/23	1600	Нильсен
3	2008/09/02	700	Хансен
4	2008/09/03	300	Хансен
5	2008/08/30	2000	Йенсен
6	2008/10/04	100	Нильсен

Теперь мы хотим найти сумму всех полей

"OrderPrice". Мы используем следующий оператор

SQL:

SELECT SUM(OrderPrice) AS OrderTotal FROM

Orders Результат будет выглядеть следующим

образом:

ОбщаяСуммаЗаказа
5700

Агрегатным функциям часто требуется добавление предложения GROUP BY.

Оператор GROUP BY

Предложение GROUP BY используется вместе с агрегатными функциями для группировки результирующего набора по одному или нескольким столбцам.

Синтаксис SQL GROUP BY
SELECT column_name,
aggregate_function(column_name) FROM table_name
WHERE column_name operator value
GROUP BY column_name

Пример SQL GROUP BY

У нас есть следующая таблица «Заказы»:

O_Id	ДатаЗаказа	ЦенаЗаказа	Клиент
1	2008/11/12	1000	Хансен
2	2008/10/23	1600	Нильсен
3	2008/09/02	700	Хансен
4	2008/09/03	300	Хансен
5	2008/08/30	2000	Йенсен
6	2008/10/04	100	Нильсен

Теперь мы хотим найти общую сумму (общий заказ) для каждого клиента.

Нам нужно будет использовать оператор GROUP BY, чтобы

сгруппировать клиентов. Мы используем следующий оператор

SQL:

SELECT Customer,SUM(OrderPrice) FROM Orders
GROUP BY Customer

Результирующий набор будет выглядеть так:

Клиент	СУММА(ЦенаЗаказа)
Хансен	2000
Нильсен	1700

Йенсен	2000

Отлично! Не так ли? :)

Давайте посмотрим, что произойдет, если мы опустим предложение GROUP BY:

SELECT Customer,SUM(OrderPrice) FROM Orders

Результат будет выглядеть следующим образом:

Клиент	СУММА(ЦенаЗаказа)
Хансен	5700
Нильсен	5700
Хансен	5700
Хансен	5700
Йенсен	5700
Нильсен	5700

Результат выше не тот, который мы хотели.

Объяснение, почему приведенный выше оператор SELECT не может быть использован:В приведенном выше операторе SELECT указаны два столбца (Customer и SUM(OrderPrice)). "SUM(OrderPrice)" возвращает одно значение (то есть общую сумму столбца "OrderPrice"), в то время как "Customer" возвращает 6 значений (по одному значению для каждой строки в таблице "Orders"). Поэтому это не даст нам правильного результата. Однако вы видели, что предложение GROUP BY решает эту проблему.

Группировка GROUP BY по нескольким столбцам

Мы также можем использовать предложение GROUP BY для нескольких

столбцов, например: SELECT Customer,OrderDate,SUM(OrderPrice) FROM

Orders
GROUP BY Customer,OrderDate

Функция UCASE()

Функция UCASE() преобразует значение поля в верхний

регистр. Синтаксис SQL UCASE()
SELECT UCASE(column_name) FROM table_name

Синтаксис для SQL Server
SELECT UPPER(column_name) FROM table_name

Пример использования SQL UCASE()

У нас есть следующая таблица "Persons":

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер

Теперь мы хотим выбрать содержимое столбцов "LastName" и "FirstName" выше и преобразовать столбец "LastName" в верхний регистр.

Мы используем следующий оператор SELECT:

SELECT UCASE(LastName) as LastName,FirstName FROM Persons

Результат будет выглядеть следующим образом:

Фамилия	Имя
ХАНСЕН	Оля
СВЕНДСОН	Туве
ПЕТТЕРСЕН	Кари

Функция LCASE()

Функция LCASE() преобразует значение поля в нижний

регистр. Синтаксис SQL LCASE()
SELECT LCASE(column_name) FROM table_name

Синтаксис для SQL Server
SELECT LOWER(column_name) FROM table_name

Пример LCASE() в SQL

У нас есть следующая таблица "Persons":

P_Id	Фамилия	Имя	Адрес	Город

1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер

Теперь мы хотим выбрать содержимое столбцов "LastName" и "FirstName" выше и преобразовать столбец "LastName" в нижний регистр.

Мы используем следующий оператор SELECT:

SELECT LCASE(LastName) as LastName,FirstName FROM Persons

Результат будет выглядеть следующим образом:

Фамилия	Имя
hansen	Оля
svendson	Туве
pettersen	Кари

Функция MID()

Функция MID() используется для извлечения символов из

текстового поля. Синтаксис SQL MID()
SELECT MID(column_name,start[,length]) FROM table_name

Параметр	Описание
имя_столбца	Обязательно. Поле, из которого нужно извлечь символы
начало	Обязательно. Указывает начальную позицию (начиная с 1)
длина	Необязательно.Количество возвращаемых символов.Если опущено, функция MID() возвращает оставшуюся часть текста

Пример SQL MID()

У нас есть следующая таблица "Persons":

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн 10	Саннес
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер

Теперь мы хотим извлечь первые четыре символа из столбца "City",

указанного выше. Мы используем следующий оператор SELECT:

SELECT MID(City,1,4) as SmallCity FROM Persons

Результат будет выглядеть следующим образом:

SmallCity
Песок
Песок
Состояние

Функция LEN()

Функция LEN() возвращает длину значения в текстовом поле.

Синтаксис SQL LEN()
SELECT LEN(column_name) FROM table_name

Пример SQL LEN()

У нас есть следующая таблица "Persons":

P_Id	Фамилия	Имя	Адрес	Город
1	Хансен	Оля	Тимотеивн	Саннес

			10	
2	Свендсон	Туве	Borgvn 23	Саннес
3	Петтерсен	Кари	Storgt 20	Ставангер

Теперь мы хотим выбрать длину значений в столбце «Адрес», указанном выше. Мы используем следующий оператор SELECT:

SELECT LEN(Address) as LengthOfAddress FROM Persons

Результат будет выглядеть следующим образом:

ДлинаАдреса
12
9
9

Функция ROUND()

Функция ROUND() используется для округления числового поля до указанного количества десятичных знаков. Синтаксис SQL ROUND()

SELECT ROUND(column_name,decimals) FROM table_name

Параметр	Описание
имя_столбца	Обязательно. Поле для округления.
десятичные дроби	Обязательно. Указывает количество возвращаемых десятичных знаков.

Пример SQL ROUND()

У нас есть следующая таблица "Products":

Код_продукта	Название продукта	Модуль	Цена за единицу
1	Ярлсберг	1000 г	10.45
2	Маскарпоне	1000 г	32.56
3	Горгонзола	1000 г	15.67

Теперь мы хотим отобразить название продукта и цену, округленную до

ближайшего целого числа. Мы используем следующий оператор SELECT:

SELECT ProductName, ROUND(UnitPrice,0) as UnitPrice FROM Products

Результат будет выглядеть следующим образом:

Название продукта	Цена за единицу
Ярлсберг	10
Маскарпоне	33
Горгонзола	16

Функция NOW()

Функция NOW() возвращает текущую системную дату и

время. Синтаксис SQL NOW()
SELECT NOW() FROM table_name

Пример SQL NOW()

У нас есть следующая таблица "Products":

Код_продукта	Название продукта	Модуль	Цена за единицу
1	Ярлсберг	1000 г	10.45
2	Маскарпоне	1000 г	32.56
3	Горгонзола	1000 г	15.67

Теперь мы хотим отобразить продукты и цены на текущую

дату. Мы используем следующий оператор SELECT:

SELECT ProductName, UnitPrice, Now() as PerDate FROM Products

Результат будет выглядеть следующим образом:

Название продукта	Цена за единицу	ЗаДату
Ярлсберг	10.45	10/7/2008 11:25:02
Маскарпоне	32.56	10/7/2008 11:25:02
Горгонзола	15.67	10/7/2008 11:25:02

Функция FORMAT()

Функция FORMAT() используется для форматирования отображения

поля. Синтаксис SQL FORMAT()
SELECT FORMAT(column_name,format) FROM table_name

Параметр	Описание
имя_столбца	Обязательно. Форматируемое поле.
формат	Обязательно. Задает формат.

Пример SQL FORMAT()

У нас есть следующая таблица "Products":

Код_продукта	Название продукта	Модуль	Цена за единицу
1	Ярлсберг	1000 г	10.45
2	Маскарпоне	1000 г	32.56
3	Горгонзола	1000 г	15.67

Теперь мы хотим отобразить продукты и цены на текущую дату (с отображением текущей даты в следующем формате "YYYY-MM-DD").

Мы используем следующий оператор SELECT:

```
SELECT ProductName, UnitPrice, FORMAT(Now(),'YYYY-MM-DD') as
PerDate FROM Products
```

Результирующий набор будет выглядеть так:

Название продукта	Цена за единицу	ЗаДату
Ярлсберг	10.45	2008-10-07
Маскарпоне	32.56	2008-10-07
Горгонзола	15.67	2008-10-07