Школа

SQL

Данная серия документов подготовлена на основе материалов сайта Школы Консорциума W3C. Этот сайт является экспериментальным сервером, на котором содержание документов хранится в формате XML. Пользователям сайта эти документы доступны в виде HTML (преобразование на строне клиента с помощью таблицы стилей XSLT) и в виде PDF (преобразование тех же документов в XSL-FO, а затем в формат PDF).

Добро пожаловать в школу SQL

Школа SQL

SQL - это язык, являющийся стандартом ANSI, предназначенный для работы с базами данных. На школе SQL вы узнаете, как применять SQL для получения доступа, задания и манипуляции с различными видами баз данных, например, Oracle, DB2, Sybase, Informix, Microsoft SQL Server, Access, и другими. Изучайте SQL!

Тест на знание SQL

Проверьте свое знание SQL! Пройдите тест на знание SQL!

Книги по SQL

Ищите хорошую книгу по SQL? Почитайте наш обзор!

Основы SQL: содержание

Введение в SQL[1] (open link)

Что такое SQL и как он используется.

SELECT[2] (open link)

Как в SQL применяется выражение SELECT для выбора данных из таблицы.

WHERE[3] (open link)

Как использовать оператор WHERE для задания критериев отбора данных.

AND и OR[4] (open link)

Как использовать выражения AND и OR для соединения нескольких условий в операторе WHERE.

BETWEEN[5] (open link)

Как использовать выражение BETWEEN....AND для того, чтобы найти данные из определенного диапазона.

1: http://xml.nsu.ru/sql/sql_intro.xml
2: http://xml.nsu.ru/sql/sql_select.xml
3: http://xml.nsu.ru/sql/sql_where.xml
4: http://xml.nsu.ru/sql/sql_and_or.xml
5: http://xml.nsu.ru/sql/sql_between.xml

DISTINCT[6] (open link)

Как использовать ключевое слово DISTINCT для получения из колонки только различающихся значений.

ORDER BY[7] (open link)

Как использовать ключевое слово ORDER BY для получения рядов в заданном порядке.

Упражнения[8] (open link)

Проверьте свои знания SQL.

INSERT[9] (open link)

Как используется выражение INSERT для того, чтобы в таблицу вставить новый ряд.

UPDATE[10] (open link)

Как используется выражение UPDATE для обновления или изменения рядов в таблице.

DELETE[11] (open link)

Как используется выражение DELETE для удаления рядов в таблице.

COUNT[12] (open link)

В этой главе объясняется использование встроенных в SQL функций COUNT.

Вторая ступень SQL: содержание

Функции SQL[13] (open link)

В этой главе объясняется использование встроенных в SQL функций.

GROUP BY и HAVING[14] (open link)

В этой главе объясняется использование встроенной в SQL функции GROUP BY.

Псевдоимена (alias)[15] (open link)

В этой главе объясняется, как используются псевдоимена колонок и таблиц.

Объединение[16] (open link)

Как использовать выбор информации более, чем из одной таблицы.

Создание и удаление[17] (open link)

Как создавать и удалять базы данных и таблицы.

ALTER TABLE[18] (open link)

Как используется выражение ALTER TABLE для добавления или удаления колонок в имеющейся таблице.

Тест назнание SQL

6: http://xml.nsu.ru/sql/sql_distinct.xml
7: http://xml.nsu.ru/sql/sql_orderby.xml
8: http://xml.nsu.ru/sql/sql_tryit.xml
9: http://xml.nsu.ru/sql/sql_insert.xml
10: http://xml.nsu.ru/sql/sql_update.xml
11: http://xml.nsu.ru/sql/sql_delete.xml
12: http://xml.nsu.ru/sql/sql_count.xml
13: http://xml.nsu.ru/sql/sql_functions.xml
14: http://xml.nsu.ru/sql/sql_groupby.xml
15: http://xml.nsu.ru/sql/sql_alias.xml
16: http://xml.nsu.ru/sql/sql_join.xml
17: http://xml.nsu.ru/sql/sql_create.xml
18: http://xml.nsu.ru/sql/sql_alter.xml

Тест назнание SQL[19] (open link)

Проверьте свое знание SQL!

Ресурсы по SQL

Книги по SQL[20] (open link)

Ищете хорошую книгу по SQL? Почитайте наше обозрение!

Введение в SQL

SQL - это язык, являющийся стандартом ANSI, предназначенный для работы с базами данных

Что такое SQL?

- SQL расшифровывается как язык структурированных запросов (Structured Query Language).
- SQL позволяет работать с базой данных.
- SQL - это язык, являющийся стандартом ANSI.
- SQL позволяет выполнять запросы к базе данных.
- SQL позволяет извлекать данные из базы.
- SQL позволяет вставлять в базу данных новые записи.
- SQL позволяет удалять записи из базы данных.
- SQL позволяет обновлять записи в базе данных.
- SQL очень легко изучить.

SQL является стандартом

SQL является стандартом ANSI (American National Standards Institute) для систем доступа к базам данных. Выражения SQL применяются для получения и обновления данных в базе данных.

SQL работает с такими базами данных, как Access, DB2, Informix, Microsoft SQL Server, Oracle, Sybase и многими другими (к сожалению, большинство из этих систем имеют свои собственные расширения этого языка).

Таблицы

Базы данных содержат объекты, которые называются таблицами.

В этих таблицах содержатся записи, в которых хранятся данные. Таблицы идентифицируются по именам (например "Persons (персоны)", "Orders (заказы)", "Suppliers (поставщики)").

Таблицы состоят из колонок и рядов с данными. Ряды содержат записи (например, одна запись для каждого человека). Колонки содержат данные (например, First Name (фамилия), Last Name (имя), Address (адрес), и City (город)).

19: http://www.w3schools.com/sql/sql_quiz.asp
20: http://www.w3schools.com/sql/sql_books.asp

Вот пример таблицы, имеющей имя "Persons (персоны)":

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

LastName, FirstName, Address, и City - это колонки таблицы. Ряды содержат 3 записи с информацией о 3 персонах.

Запросы SQL

Посредством SQL мы можем делать запросы к базе данных и получать результат в табличной форме.

С помощью следующего запроса:

```
SELECT LastName FROM Persons
```

Мы получим следующий результат:

LastName
Hansen
Svendson
Pettersen

Обратите внимание: в некоторых системах управления базами данных в конце каждого SQL-выражения требуется ставить точку с запятой. На нашей школе мы обходися без этого.

Управление данными в SQL

Как подсказывает название этого языка, он предназначен для выполнения запросов. Но кроме того, язык SQL содержит синтаксис для обновления записей, вставки новых записей и удаления существующих.

Вместе команды запросов и обновления образуют язык управления данными DML (Data Manipulation Language), который является частью SQL:

- SELECT - извлекает данные из базы.
- UPDATE - обновляет данные в базе.
- DELETE - удаляет данные из базы.
- INSERT - вставляет новые данные в базу.

Определение данных в SQL

Язык определения данных DDL (Data Definition Language) является частью SQL, которая позволяет создавать и удалять таблицы в базе данных. Кроме того, мы можем задавать индексы (ключи), описывать связи между таблицами и накладывать условия на таблицы в базе данных.

Наиболее важными выражениями языка DDL в SQL являются:

- CREATE TABLE - создает новую таблицу в базе данных.
- ALTER TABLE - изменяет таблицу в базе данных.
- DROP TABLE - удаляет таблицу из базы данных.
- CREATE INDEX - создает индекс (поисковый индекс).
- DROP INDEX - удаляет индекс.

SQL и ASP (Active Server Pages)

SQL является важной частью ASP, потому что ADO (Active Data Object), который применяется в ASP для работы с базами данных, основывается на SQL.

Выражение SELECT

Выражение SELECT отбирает колонки данных в базе
Табулированный результат запроса сохраняется в таблице результата (она называется результирующим множеством)

Выражение SELECT

Выражение SELECT выбирает определенные колонки данных в базе.

Оно используется по следующему шаблону:

```
SELECT имя_колонки(имена_колонок) FROM имя_таблицы
```

Пример: выбор колонок в таблице

Чтобы выбрать колонки "LastName (фамилия)" и "FirstName (имя)", следует использовать примерно такое выражение SELECT:

```
SELECT LastName,FirstName FROM Persons
```

Таблица Persons (персоны):

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

Результат:

LastName	FirstName
Hansen	Ola
Svendson	Tove
Pettersen	Kari

Пример: выбор всех колонок

Чтобы выбрать все колонки в таблице "Person", нужно использовать вместо имени выбираемой колонки символ *, вот так:

```
SELECT * FROM Persons
```

Результат запроса:

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

Таблица результата

Результат запроса SQL сохраняется в результирующем множестве. На него можно смотреть как на таблицу данных, в которой содержится результат запроса. Большинство программ работы с базами данных позволяют передвигаться по результирующему множеству посредством программных функций, например, Move-To-First-Record (перейти к первой записи), Get-Record-Content (показать содержимое записи), Move-To-Next-Record (перейти к следующей записи) и т.д.

На этой школе мы не рассматриваем подобные программные функции.

Если вы хотите узнать больше о доступе к SQL-данным посредством вызова функций, посетите нашу школу ADO.

Использовать ли точку с запятой после выражений SQL?

Во многих руководствах по SQL настаивают, чтобы вы всегда завершали выражение SQL точкой с запятой.

Это не правильно. Но точка с запятой является стандартным способом завершения выражения SQL (и разделения нескольких выражений) в системах, в которых позволяется выполнение нескольких выражений SQL в течении одного обращения к серверу базы данных.

Оператор WHERE

Оператор WHERE применяется для задания критериев отбора

Оператор WHERE

Чтобы выбирать данные в таблице по определенному критерию, в выражение SELECT можно добавить оператор WHERE по следующему шаблону:

```
SELECT имя_колонки FROM имя_таблицы
WHERE имя_колонки условие значение
```

С оператором WHERE можно использовать следующие условия:

Оператор	Условие
=	Равно
<>	Не равно
>	Больше чем
<	Меньше чем
>=	Больше или равно
<=	Меньше или равно
BETWEEN	Принадлежит заданному промежутку, включая его края
LIKE	Объясняется ниже

Обратите внимание: в некоторых вестиях SQL оператор <> (не равно) может быть записан как !=.

Пример: выбор персон (persons) из определенного города (city)

Чтобы выбрать из таблицы только записи о персонах, которые живут в городе Sandnes, следует добавить в выражение SELECT оператор WHERE, вот так:

```
SELECT * FROM Persons WHERE City='Sandnes'
```

Таблица Persons (персоны):

LastName	FirstName	Address	City	Year
Hansen	Ola	Timoteivn 10	Sandnes	1951
Svendson	Tove	Borgvn 23	Sandnes	1978
Svendson	Stale	Kaivn 18	Sandnes	1980
Pettersen	Kari	Storgt 20	Stavanger	1960

Результат запроса:

LastName	FirstName	Address	City	Year
Hansen	Ola	Timoteivn 10	Sandnes	1951
Svendson	Tove	Borgvn 23	Sandnes	1978
Svendson	Stale	Kaivn 18	Sandnes	1980

Использование кавычек

Обратите внимание, что в этом примере мы значение, записанное в условие, заключили в одинарные кавычки. В SQL одинарные кавычки используется для задания текстовых значений. Большинство систем управления базами данных позволяют использовать и двойные ка-

вычки. Числовые значения не нужно заключать в кавычки.

Для текстовых значений:

```
Вот так правильно:
SELECT * FROM Persons WHERE FirstName='Tove'
Вот так не правильно:
SELECT * FROM Persons WHERE FirstName=Tove
```

Для числовых значений:

```
Вот так правильно:
SELECT * FROM Persons WHERE Year>1965
Вот так не правильно:
SELECT * FROM Persons WHERE Year>'1965'
```

Условие LIKE

Условие LIKE применяется для задания поиска в колонке по определенному шаблону (паттерну).

Синтакс используется такой:

```
SELECT имя_колонки FROM имя_таблицы
WHERE имя_колонки LIKE паттерн
```

В начале и в конце паттерна для задания любых букв в паттерне можно использовать символ "%".

Пример: выбор персоны по паттерну имени персоны

Это выражение SQL выберет всех персон, имена которых начинаются с буквы 'O':

```
SELECT * FROM Persons WHERE FirstName LIKE 'O%'
```

Это выражение SQL выберет всех персон, имена которых кончаются буквой 'a':

```
SELECT * FROM Persons WHERE FirstName LIKE '%a'
```

Это выражение SQL выберет всех персон, в именах которых имеется комбинация букв 'la':

```
SELECT * FROM Persons WHERE FirstName LIKE '%la%'
```

Во всех этих примерах мы получим результат:

LastName	FirstName	Address	City	Year
Hansen	Ola	Timoteivn 10	Sandnes	1951

Операторы AND и OR

Оператор WHERE

Операторы AND и OR позволяют соединить несколько условий в операторе WHERE.

Оператор AND позволяет выбрать ряд, если для него выполняются все перечисленные усло-

вия. Оператор OR позволяет выбрать ряд, если для него выполняется хотя бы одно из перечисленных условий.

Оригинальная таблица (используется в этих примерах)

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Svendson	Stephen	Kaivn 18	Sandnes

Пример 1

Оператор AND используется для выбора всех персон, чьи имена равны "Tove", а фамилии равны "Svendson":

```
SELECT * FROM Persons
WHERE FirstName='Tove'
AND LastName='Svendson'
```

Результат запроса:

LastName	FirstName	Address	City
Svendson	Tove	Borgvn 23	Sandnes

Пример 2

Оператор OR используется для выбора всех персон, чьи имена равны "Tove" или чьи фамилии равны "Svendson":

```
SELECT * FROM Persons
WHERE firstname='Tove'
OR lastname='Svendson'
```

Результат запроса:

LastName	FirstName	Address	City
Svendson	Tove	Borgvn 23	Sandnes
Svendson	Stephen	Kaivn 18	Sandnes

Пример 3

Кроме того, можно комбинировать операторы AND and OR (для формирования сложных выражений используются скобки):

```
SELECT * FROM Persons WHERE
(FirstName='Tove' OR FirstName='Stephen')
```

```
AND LastName='Svendson'
```

Результат запроса:

LastName	FirstName	Address	City
Svendson	Tove	Borgvn 23	Sandnes
Svendson	Stephen	Kaivn 18	Sandnes

Оператор BETWEEN ... AND

Оператор BETWEEN ... AND

Оператор BETWEEN ... AND выбирает данные в определенном диапазоне, включая его края. Значения, отмечающие края диапазона, могут быть числовыми, текстовыми или в формате даты.

```
SELECT имя_колонки FROM имя_таблицы
WHERE имя_колонки
BETWEEN значение_1 AND значение_2
```

Оригинальная таблица (используется в этих примерах)

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Nordmann	Anna	Neset 18	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

Пример 1

Для того, чтобы выбрать все персоны, фамилии которых по алфавиту находятся между "Hansen" и "Pettersen" (включая эти фамилии), используется следующий SQL-запрос:

```
SELECT * FROM Persons WHERE LastName
BETWEEN 'Hansen' AND 'Pettersen'
```

Результат запроса:

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Nordmann	Anna	Neset 18	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

Пример 2

Для того, чтобы выбрать всех персон, чьи фамилии лежат вне указанного в предыдущем примере диапазона, используется оператор NOT:

```
SELECT * FROM Persons WHERE LastName
NOT BETWEEN 'Hansen' AND 'Pettersen'
```

Результат запроса:

LastName	FirstName	Address	City
Svendson	Tove	Borgvn 23	Sandnes

SELECT DISTINCT

Ключевое слово DISTINCT используется, когда нужно получить только различающиеся значения.

Ключевое слово DISTINCT

SQL-выражение SELECT возвращает информацию из колонок таблицы. Но что делать, если мы хотим отобрать только различающиеся элементы?

В SQL для этого нужно к выражению SELECT добавить ключевое слово DISTINCT, используя следующий синтаксис:

```
SELECT DISTINCT имя(имена)_колонки FROM имя_таблицы
```

Пример: выбор компаний из таблицы заказов

Пусть мы имеем простую таблицу Orders, в которой указаны названия компаний и номера заказов:

Company	OrderNumber
Sega	3412
W3Schools	2312
Trio	4678
W3Schools	6798

Это SQL-выражениет

```
SELECT Company FROM Orders
```

даст такой результат:

Company
Sega

W3Schools
Trio
W3Schools

Обратите внимание, что имя компании W3Schools фигурирует в результате запроса не один раз. Иногда нам этого не нужно.

Пример: выбор различных компаний из таблицы заказов

А вот это SQL-выражениет

```
SELECT DISTINCT Company FROM Orders
```

даст другой результат:

Company
Sega
W3Schools
Trio

Обратите внимание, что теперь имя компании W3Schools фигурирует в результате запроса только один раз. Это то, что нужно.

Ключевое слово ORDER BY

Ключевое слово ORDER BY применяется для сортировки и упорядочивания результатов запроса

Сортировка рядов

Конструкция ORDER BY применяется для сортировки рядов.

Таблица Orders (заказы):

Company	OrderNumber
Sega	3412
ABC Shop	5678
W3Schools	2312
W3Schools	6798

Пример 1

Отображение компаний в алфавитном порядке:

```
SELECT Company, OrderNumber FROM Orders
```

```
ORDER BY Company
```

Результат запроса:

Company	OrderNumber
ABC Shop	5678
Sega	3412
W3Schools	6798
W3Schools	2312

Пример 2

Отображение компаний в алфавитном порядке и номера заказов в порядке возрастания:

```
SELECT Company, OrderNumber FROM Orders
ORDER BY Company, OrderNumber
```

Результат запроса:

Company	OrderNumber
ABC Shop	5678
Sega	3412
W3Schools	2312
W3Schools	6798

Пример 3

Отображение компаний в обратном алфавитном порядке:

```
SELECT Company, OrderNumber FROM Orders
ORDER BY Company DESC
```

Результат запроса:

Company	OrderNumber
W3Schools	6798
W3Schools	2312
Sega	3412
ABC Shop	5678

Поупражняйтесь в SQL-запросах

Здесь вы сможете поупражняться в создании SQL-запросов.

Мы будем использовать таблицу Customers в базе данных Northwind:

CompanyName	ContactName	Address	City
Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin
Antonio Moreno Taquerна	Antonio Moreno	Mataderos 2312	Мйxico D.F.
Berglunds snabbkцp	Christina Berglund	Berguvsvдgen 8	Lulee
Ernst Handel	Roland Mendel	Kirchgasse 6	Graz
North/South	Simon Crowther	South House 300 Queensbridge	London
Romero y tomillo	Alejandra Camino	Gran Vна, 1	Madrid
Simons bistro	Jytte Petersen	Vinbжltet 34	Kшbenhavn
The Big Cheese	Liz Nixon	89 Jefferson Way Suite 2	Portland
Wolski Zajazd	Zbyszek Piestrzeniewicz	ul. Filtrowa 68	Warszawa

Попробуйте различные запросы

Посмотрите, как действует SQL: скопируйте SQL-запросы, приведенные здесь и вставьте их в текстовое поле. Можно попробовать и свои собственные запросы:

```
SELECT * FROM customers

SELECT CompanyName, ContactName
FROM customers

SELECT * FROM customers
WHERE companyname LIKE 'a%'

SELECT CompanyName, ContactName
FROM customers
WHERE CompanyName > 'g'
AND ContactName > 'g'
```

- ! -

Данное интерактивное содержание документа может быть представлено только в варианте он-лайн. Он находится на http://xml.nsu.ru.

Выражение INSERT INTO

Вставка новых рядов

Выражение INSERT INTO вставляет в таблицу новые ряды:

```
INSERT INTO имя_таблицы
VALUES (значение_1, значение_2,....)
```

Кроме того, вы можете конкретизировать колонки, в которые вы хотите вставить данные:

```
INSERT INTO имя_таблицы (колонка_1, колонка_2,...)
VALUES (значение_1, значение_2,....)
```

Вставка нового ряда

Вот таблица Persons:

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger

А вот SQL-выражение:

```
INSERT INTO Persons
VALUES ('Hetland', 'Camilla', 'Hagabakka 24', 'Sandnes')
```

Мы получим вот такой результат:

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger
Hetland	Camilla	Hagabakka 24	Sandnes

Вставка данных в определенные колонки

Вот таблица Persons:

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger
Hetland	Camilla	Hagabakka 24	Sandnes

А вот SQL-выражение:

```
INSERT INTO Persons (LastName, Address)
VALUES ('Rasmussen', 'Storgt 67')
```

Мы получим вот такой результат:

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger
Hetland	Camilla	Hagabakka 24	Sandnes
Rasmussen		Storgt 67	

Выражение UPDATE

Обновление рядов

Выражение UPDATE обновляет или изменяет ряды:

```
UPDATE имя_таблицы SET имя_колонки = новое_значение
WHERE имя_колонки = некоторое_значение
```

Таблица Person

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen		Storgt 67	

Обновление одной колонки в ряде

Мы хотим добавить имя персоны, фамилия которой "Rasmussen":

```
UPDATE Person SET FirstName = 'Nina'
WHERE LastName = 'Rasmussen'
```

Мы получим вот такой результат:

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen	Nina	Storgt 67	

Обновление нескольких колонок в ряде

Теперь мы хотим изменить адрес и добавить название города:

```
UPDATE Person
SET Address = 'Stien 12', City = 'Stavanger'
WHERE LastName = 'Rasmussen'
```

Мы получим вот такой результат:

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen	Nina	Stien 12	Stavanger

Выражение DELETE

Удаление рядов

Выражение DELETE позволяет удалять ряды из таблицы:

```
DELETE FROM имя_таблицы WHERE имя_колонки = значение
```

Таблица Person

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen	Nina	Stien 12	Stavanger

Удаление ряда

Удалим запись о персоне Nina Rasmussen:

```
DELETE FROM Person WHERE LastName = 'Rasmussen'
```

Мы получим вот такой результат:

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger

Функции COUNT

В SQL имеются встроенные функции для подсчета количества записей в базе данных

Синтаксис функции подсчета

Синтаксис встроенных функций COUNT выглядит так:

```
SELECT COUNT(колонка) FROM таблица
```

Функция COUNT(*)

Функция COUNT(*) возвращает число выбранных рядов в результате запроса.

Пусть у нас имеется таблица Persons:

Name	Age
Hansen, Ola	34
Svendson, Tove	45
Pettersen, Kari	19

Этот запрос возвратит число рядов в таблице:

```
SELECT COUNT(*) FROM Persons
```

Мы получим такой результат:

```
3
```

А этот запрос возвратит число персон, которым больше 20 лет:

```
SELECT COUNT(*) FROM Persons WHERE Age>20
```

Мы получим такой результат:

```
2
```

Функция COUNT(колонка)

Функция COUNT(колонка) возвращает число рядов в заданной колонке, чье значение в этой колонке не равно пустому (NULL).

Пусть у нас имеется таблица Persons:

Name	Age
Hansen, Ola	34
Svendson, Tove	45
Pettersen, Kari	

В результате этого запроса мы найдем количество персон, у которых в поле "Age" таблицы "Persons" выставлено какое-нибудь значение:

```
SELECT COUNT(Age) FROM Persons
```

Мы получим такой результат:

```
2
```

Функция COUNT(column) годится для поиска колонок, в которых не выставлено какое-то значение. Обратите в этом примере внимание на то, что результат меньше на единицу, чем общее число колонок в таблице, потому что у одной из персон значение возраста отсутствует.

COUNT DISTINCT

В выражении COUNT можно использовать ключевое слово DISTINCT и тогда будут подсчитаны только различающиеся результаты.

Синтаксис при этом такой:

```
SELECT DISTINCT COUNT(колонка(колонки)) FROM таблица
```

Пусть у нас имеется таблица Orders:

Company	OrderNumber
Sega	3412
W3Schools	2312
Trio	4678
W3Schools	6798

При запросе SQL:

```
SELECT COUNT(Company) FROM Orders
```

Мы получим такой результат:

```
4
```

А вот с этим запросом SQL:

```
SELECT DISTINCT COUNT(Company) FROM Orders
```

Мы получим такой результат:

```
3
```

Функции SQL

В SQL имеется множество встроенных функций для подсчетов и вычислений

Синтаксис функций

Синтаксис встроенных в SQL функций таков:

```
SELECT function(колонка) FROM таблица
```

Оригинальная таблица (используется в примерах)

Name	Age
Hansen, Ola	34
Svendson, Tove	45
Pettersen, Kari	19

Функция AVG(колонка)

Функция AVG возвращает среднее значение колонки в выбранной области. Пустые значения не учитываются в подсчете среднего.

Пример 1

В этом примере возвращается среднее значение возраста персон из таблицы Persons:

```
SELECT AVG(Age) FROM Persons
```

Мы получим такой результат:

```
32.67
```

Пример 2

В этом примере возвращается среднее значение возраста персон из таблицы Persons, которые старше 20 лет:

```
SELECT AVG(Age) FROM Persons where Age>20
```

Мы получим такой результат:

```
39.5
```

Функция MAX(колонка)

Функция MAX возвращает максимальное значение колонки. Пустые значения не учитываются.

Пример

```
SELECT MAX(Age) FROM Persons
```

Мы получим такой результат:

```
45
```

Функция MIN(колонка)

Функция MIN возвращает минимальное значение колонки. Пустые значения не учитываются.

Пример

```
SELECT MIN(Age) FROM Persons
```

Мы получим такой результат:

```
19
```

Обратите внимание: Функции MIN и MAX можно использовать и в текстовых колонках, чтобы найти первые или последние значения по алфавитному порядку.

Функция SUM(колонка)

Функция SUM возвращает общую сумму значений в колонке в выбранной области. Пустые значения в вычислениях не учитываются.

Пример 1

В этом примере мы получим суммарный возраст персон из таблицы Person:

```
SELECT SUM(Age) FROM Persons
```

Мы получим результат:

```
98
```

Пример 2

В этом примере мы получим суммарный возраст персон из таблицы Person, чей возраст больше 20 лет:

```
SELECT SUM(Age) FROM Persons where Age>20
```

Мы получим результат:

```
79
```

Ключевые слова GROUP BY и HAVING

Функциям агрегирования (например, SUM) часто необходима дополнительная функциональность, которая добавляется ключевым словом GROUP BY

Ключевые слова GROUP BY

Ключевые слова GROUP BY были добавлены в SQL, поскольку функции агрегирования

(например, SUM) возвращают комбинацию значений всей колонки каждый раз, когда вызывается эта функция.

Без дополнительной функциональности, которую добавляет ключевое слово GROUP BY, например, невозможно найти сумму значений в какой-либо группе значений в колоке.

Синтаксис ключевого слова GROUP BY таков:

```
SELECT колонка,SUM(колонка)
FROM таблица GROUP BY колонка
```

Пример использования ключевого слова GROUP BY

Вот таблица продаж Sales:

Company	Amount
W3Schools	5500
IBM	4500
W3Schools	7100

При запросе:

```
SELECT Company, SUM(Amount) FROM Sales
```

Мы получим такой результат:

Company	SUM(Amount)
W3Schools	17100
IBM	17100
W3Schools	17100

Приведенный нами пример кода запроса не верен, поскольку возвращенная колонка не является частью комбинации-агрегата. Используя ключевое слово GROUP BY исправит ошибку, как в этом SQL-запросе:

```
SELECT Company,SUM(Amount) FROM Sales
GROUP BY Company
```

Результат:

Company	Amount
W3Schools	12600
IBM	4500

Ключевое слово HAVING

Ключевое слово HAVING было добавлено в SQL поскольку ключевое слово WHERE не может быть использовано совместно с функциями агрегирования (например, SUM).

Без ключевого слова HAVING было бы невозможно проверить значения, получаемые в ре-

зультате действия функции, на какое-либо условие.

Синтакис ключевого слова HAVING таков:

```
SELECT колонка,SUM(колонка) FROM таблица
GROUP BY колонка
HAVING SUM(колонка) условие значение
```

Вот таблица продаж Sales:

Company	Amount
W3Schools	5500
IBM	4500
W3Schools	7100

При запросе:

```
SELECT Company,SUM(Amount) FROM Sales
GROUP BY Company HAVING SUM(Amount)>10000
```

Мы получим такой результат:

Company	SUM(Amount)
W3Schools	12600

Псевдоимена (alias)

В SQL для имен таблиц и колонок можно использовать псевдоимена

Псевдоимя колонки

Синтаксис таков:

```
SELECT колонка AS псевдоимя_колонки FROM таблица
```

Псевдоимя таблицы

Синтаксис таков:

```
SELECT колонка FROM таблица AS псевдоимя_таблицы
```

Пример: использование псевдоимени колонки

Вот таблица Persons:

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

А вот запрос SQL

```
SELECT LastName AS Family, FirstName AS Name
FROM Persons
```

Результат:

Family	Name
Hansen	Ola
Svendson	Tove
Pettersen	Kari

Пример: использование псевдоимени таблицы

Вот таблица Persons:

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

А вот запрос SQL

```
SELECT LastName, FirstName
FROM Persons AS Employees
```

Результат:

Таблица Employees:

Family	Name
Hansen	Ola
Svendson	Tove
Pettersen	Kari

Объединение

Объединения и ключи

Иногда для получения результата нам нужно объединить две таблицы. Тогда нам следует выполнить объединение.

Таблицы в базе данных могут быть соотнесены друг с другом с помощью ключей. Первичный ключ (primary key) - это колонка, в которой содержится уникальное значение для каждого ряда. Ее назначение - помочь связать данные из разных таблиц вместе без полного дублирования всех данных из обоих таблиц.

В приведенной ниже таблице Employees, колонка ID является первичным ключом, это означает, что два различных ряда не могут иметь одинаковый ID. Идентификатор ID различает двух персон, даже если у них одинаковые имена.

Взглянув на пример таблицы, приведенный ниже, вы заметите, что:

- Колонка ID является первичным ключом таблицы Employees.
- Колонка ID в таблице Orders используется для обращения к персонам из таблицы Employees без использования их имен.

Оригинальные таблицы (используются в примерах)

Вот таблица Employees:

ID	Name
01	Hansen, Ola
02	Svendson, Tove
03	Svendson, Stephen
04	Pettersen, Kari

Вот таблица Orders:

ID	Product
01	Printer
02	Table
03	Chair

Обращение к двум таблицам

Вы можете выбирать данные из двух таблиц, обращаясь к двум таблицам, вот так:

Пример 1

Кто заказал продукт и что именно было заказано?

```
SELECT Employees.Name, Orders.Product
FROM Employees, Orders
WHERE Employees.ID = Orders.ID
```

Результат:

Name	Product
Hansen, Ola	Printer
Svendson, Stephen	Table
Svendson, Stephen	Chair

Пример 2

Кто заказал принтер?

```
SELECT Employees.Name
FROM Employees, Orders
WHERE Employees.ID = Orders.ID
AND Orders.Product = 'Printer'
```

Результат:

Name
Hansen, Ola

Использование объединений

Также мы можем выбирать данные из двух таблиц, используя ключевое слово JOIN, вот так (лучше делать именно так):

Пример INNER JOIN

Синтаксис:

```
SELECT поле_1, поле_2, поле_3
FROM первая_таблица
INNER JOIN вторая_таблица
ON первая_таблица.ключевое_поле
= вторая_таблица.внешнее_ключевое_поле
```

Кто заказал продукт и какой именно?

```
SELECT Employees.Name, Orders.Product
FROM Employees
INNER JOIN Orders
ON Employees.ID = Orders.ID
```

INNER JOIN возвращает все ряды из обоих таблиц, в которых имеется соответствие. Если в таблице Employees есть ряды, которые не имеют соответствия в таблице Orders, эти ряды не выдаются в результат.

Результат:

Name	Product
Hansen, Ola	Printer
Svendson, Stephen	Table
Svendson, Stephen	Chair

Пример LEFT JOIN

Синтаксис:

```
SELECT поле_1, поле_2, поле_3
FROM первая_таблица
LEFT JOIN вторая_таблица
ON первая_таблица.ключевое_поле
= вторая_таблица.внешнее_ключевое_поле
```

Перечислить всех работников и их заказы - если есть.

```
SELECT Employees.Name, Orders.Product
FROM Employees
LEFT JOIN Orders
ON Employees.ID = Orders.ID
```

LEFT JOIN возвращает все ряды из первой таблицы Employees, даже если какому-то ряду нет соответствия в таблице Orders. Даже если в таблице Employees есть ряды, для которых в таблице Orders нет соответствий, они все равно будут выведены в результат.

Результат:

Name	Product
Hansen, Ola	Printer
Svendson, Tove	
Svendson, Stephen	Table
Svendson, Stephen	Chair
Pettersen, Kari	

Пример RIGHT JOIN

Синтаксис:

```
SELECT поле_1, поле_2, поле_3
FROM первая_таблица
RIGHT JOIN вторая_таблица
ON первая_таблица.ключевое_поле
= вторая_таблица.внешнее_ключевое_поле
```

Перечислить все заказы и кто именно заказал - если заказал.

```
SELECT Employees.Name, Orders.Product
FROM Employees
RIGHT JOIN Orders
ON Employees.ID = Orders.ID
```

RIGHT JOIN возвращает все ряды из таблицы Orders, даже если им нет соответствий в таблице Employee. Если в таблице Orders обнаруживаются ряды, которым в таблице Employees нет соответствий, они все равно выводятся в результат.

Результат:

Name	Product
Hansen, Ola	Printer
Svendson, Stephen	Table
Svendson, Stephen	Chair

Пример 2

Кто заказал принтер?

```
SELECT Employees.Name
FROM Employees
```

```
INNER JOIN Orders
ON Employees.ID = Orders.ID
WHERE Orders.Product = 'Printer'
```

Результат:

Name
Hansen, Ola

Создание базы данных и таблицы

Создание базы данных

Для создания базы данных используется следующий синтаксис:

```
CREATE DATABASE имя_базы_данных
```

Создание таблицы

Для создания таблицы в базе данных используется следующий синтаксис:

```
CREATE TABLE имя_таблицы
(
имя_колонки_1 тип_данных,
имя_колонки_2 тип_данных,
.......
)
```

Пример создания таблицы

В этом примере показано создание таблицы, имеющей имя Person, в таблице четыре колонки. Имена колонок соответственно: "LastName", "FirstName", "Address", и "Age":

```
CREATE TABLE Person
(
LastName varchar,
FirstName varchar,
Address varchar,
Age int
)
```

В этом примере демонстрируется, как задать максимальную длину данных в колонке:

```
CREATE TABLE Person
(
LastName varchar(30),
FirstName varchar,
Address varchar,
Age int(3)
)
```

Типы данных

Тип данных определяет, данные какого типа могут содержаться в колонке. В этой таблице перечислены самые распространенные типы данных в SQL:

Тип данных	Описание
`integer(size) int(size) smallint(size) tinyint(size)`	Хранение целых чисел. Максимальное число цифр целого числа задается в скобках
`decimal(size, d) numeric(size,d)`	Хранение десятичных дробей. Максимальное число цифр в числе задается параметром "size". Максимальное число цифр справа от десятичного разделителя задается параметром "d"
`char(size)`	Хранение строк фиксированной длины (строки могут состоять из цифр, букв и специальных символов). Фиксированная длина задается в скобках
`varchar(size)`	Хранение строк переменной длины (строки могут состоять из цифр, букв и специальных символов). Максимальный размер задается в скобках
`date(yyyymmdd)`	Хранение дат

Создание индекса

Индексы создаются в таблицах для более быстрой и эффективной локализации рядов. Можно создать индексы на одну или несколько колонок таблицы, каждому индексу дается имя. Пользователи не могут видеть индексов, они используются только для повышения скорости выполнения запросов.

Обратите внимание: обновление таблицы, содержащей индексы занимает больше времени, чем не содержащей индексов, потому что индексы тоже нужно обновлять. Так что индексы хорошо создавать только для тех колонок, которые часто используются для поиска.

Уникальный индекс

Создание уникального индекса таблицы. Это означает, что два различных ряда таблицы не могут иметь одинаковое значение индекса:

```
CREATE UNIQUE INDEX имя_индекса
ON имя_таблицы (имя_колонки)
```

Имя_колонки определяет имя колоки, которую мы индексируем.

Простой индекс

Создание простого индекса таблицы. Если опустить ключевое слово UNIQUE, повторяющиеся значения индекса допускаются:

```
CREATE INDEX имя_индекса
ON имя_таблицы (имя_колонки)
```

Имя_колонки определяет имя колоки, которую мы индексируем.

Пример создания индекса

В этом примере создается простой индекс, которому дается имя PersonIndex. Индекс созда-

ется на поле LastName таблицы Person:

```
CREATE INDEX PersonIndex
ON Person (LastName)
```

Если вы хотите проиндексировать значения в колонке в порядке убывания, после имени колонки следует добавить зарезервированное слово DESC:

```
CREATE INDEX PersonIndex
ON Person (LastName DESC)
```

Если вы хотите проиндексировать несколько колонок таблицы, вы можете перечислить имена индексируемых колонок в скобках, разделяя их запятыми:

```
CREATE INDEX PersonIndex
ON Person (LastName, FirstName)
```

Удаление индекса

Можно удалить имеющийся в таблице индекс с помощью выражения DROP:

```
DROP INDEX имя_таблицы.имя_индекса
```

Удаление базы данных или таблицы

Удаление базы данных:

```
DROP DATABASE имя_базы_данных
```

Удаление таблицы:

```
DROP TABLE имя_таблицы
```

Удаление данных из таблицы с сохранением таблицы:

```
DELETE TABLE имя_таблицы
```

Выражение ALTER TABLE

ALTER TABLE

Выражение ALTER TABLE применяется для добавления или удаления колонок в имеющейся таблице:

```
ALTER TABLE имя_таблицы
ADD имя_колонки тип_данных

ALTER TABLE имя_таблицы
DROP имя_колонки
```

Обратите внимание: некоторые системы баз данных не позволяют удалять колонки из таблицы базы данных (DROP имя_колонки).

Исходная таблица Person:

LastName	FirstName	Address
Pettersen	Kari	Storgt 20

Пример добавления колонки

Добавим колонку с именем City в таблицу Person:

```
ALTER TABLE Person ADD City varchar(30)
```

Результат:

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	

Пример удаления колонки

Теперь удалим колонку с именем Address в таблице Person:

```
ALTER TABLE Person DROP Address
```

Результат:

LastName	FirstName	City
Pettersen	Kari	