Справочник по MySQL

Реляционная БД — набор взаимосвязанных данных (англ. — relation – связь).
SQL (structured query language — «язык структурированных запросов») – ЯП для управления данными в реляционной базе данных.
MySQL – реляционная система управления базами данных (программа, которая понимает SQL).

Типы данных

Наиболее часто используемые типы данных:

ЧИСЛОВЫЕ ТИПЫ
INT(M): от -2147483648 до 2147483647 или от 0 до 4294967295 UNSIGNED*. Можно определить длину отображения (M).
FLOAT (M, D): число с плавающей запятой одиночной точности - использует 4 байта для хранения значений.
Можно определить (M) и количество десятичных знаков (D).
DOUBLE (M, D): число с плавающей запятой двойной точности - использует 8 байт для хранения значений. Можно определить (M) и (D).

ДАТА И ВРЕМЯ
DATE: дата в формате ГГГГ-ММ-ДД .
DATETIME: комбинация даты и времени в формате ГГГГ-ММ-ДД ЧЧ:ММ:СС.
TIMESTAMP: отметка времени, рассчитанная с полуночи, 1 января 1970 г.
TIME: хранит время в формате ЧЧ:ММ:СС.

СТРОКОВЫЕ ТИПЫ
CHAR (M): строка символов с фиксированной длиной. Размер указывается в скобках от 1 до 8000 символов.
Данные CHAR при хранении дополняются справа пробелами до заданной длины. Эти пробелы удаляются при извлечении данных.
VARCHAR (M): строка символов переменной длины. Размер указывается в скобках от 1 до 8000 символов.
При хранении данных VARCHAR используется только то количество символов, которое необходимо.
BLOB: «Binary Large Objects» используются для хранения больших объемов двоичных данных, например изображений или других типов файлов.
TEXT: большое количество текстовых данных.

Что использовать CHAR / VARCHAR?
- От 1 до 3 символов => *char*
- От 4 до 7 символов => если большая часть ваших данных имеет 7 символов - *char*.
- Более 7 символов => varchar. Если, конечно, ваши данные не имеют фиксированной длины, которая никогда не меняется.

Value	CHAR(4)	Storage Required	VARCHAR(4)	Storage Required
''	' '	4 bytes	''	1 byte
'ab'	'ab '	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	5 bytes
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

* Целочисленные типы имеют дополнительный параметр UNSIGNED. Обычно целое число переходит от отрицательного к положительному. Добавление атрибута UNSIGNED будет перемещать этот диапазон вверх так, чтобы он начинался с нуля вместо отрицательного числа.
Полный список: https://www.w3schools.com/sql/sql_datatypes.asp

CREATE DATABASE – Создание, выбор и удаление базы данных

Создание БД с именем tests:
```
CREATE DATABASE `tests`;
```

Создание БД с именем tests и установка кодировки:
```
CREATE DATABASE `tests`
DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
```

Выбор среди списка баз данных базы с именем tests:
```
USE `tests`;
```

УДАЛИТЬ базу данных:
```
DROP DATABASE `tests`;
```

PRIMARY KEY / FOREIGN KEY - Первичный ключ и внешний ключ

Первичный ключ – идентификатор строки, имеет уникальное значение, не должен меняться со временем, не может содержать NULL.
Ключевое слово PRIMARY KEY указывает, какое поле является первичным ключем.
За уникальное значение первичного ключа отвечает атрибут AUTO_INCREMENT.
Внешний ключ (или ссылочный) – это ключ, который используется для связи двух таблиц.
Значение внешнего ключа одной таблицы соответствует значению первичного ключа другой таблицы.
Ключевое слово FOREIGN KEY указывает, какое поле является внешним ключем.

Таблица authors:

id	author
1	a
2	b
3	c

Таблица books:

authorId	bookName
3	а
2	б
3	в
1	г

В таблице books поле authorId является Внешним ключом и ссылается на поле id таблицы authors.
Для создания таблиц с такой структурой используется следующий код:

```
CREATE TABLE authors (
        id int NOT NULL PRIMARY KEY,          // id - первичный ключ
        author varchar(50)
);
CREATE TABLE books (
        authorId int FOREIGN KEY REFERENCES authors(id),          // authorId - внешний ключ
        bookName varchar(100)
);
```

CREATE TABLE - Создание таблицы

При создании таблицы необходимо назвать таблицу, определить её столбцы и тип данных каждого столбца.
Создание таблицы customers с 4 столбцами: «id», «firstname», «email» и «reg_date»

```
CREATE TABLE `customers` (
  id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  firstname VARCHAR(30) NOT NULL,
  email VARCHAR(50),
  reg_date TIMESTAMP
);
```

Примечания к таблице выше на примере столбца firstname:
firstname VARCHAR(30) NOT NULL, где -
  * firstname - имя столбца;
  * VARCHAR(30) - тип данных, которые может содержать столбец;
  * Другие необязательные атрибуты (АТРИБУТЫ ОГРАНИЧЕНИЙ) для каждого столбца:
NOT NULL - каждая строка должна содержать значение, нулевые значения не допускаются;
DEFAULT value - установить значение по умолчанию, которое добавляется, когда другое значение не передается;
UNSIGNED - используется для чисел, ограничивает сохраненные данные положительными числами и нолем;
AUTO_INCREMENT - MySQL автоматически увеличивает значение поля на 1 каждый раз при добавлении новой записи;
PRIMARY KEY - используется для уникальной идентификации строк в таблице.
Столбец с параметром PRIMARY KEY часто является идентификационным номером и часто используется с AUTO_INCREMENT.

Добавление, удаление, правка столбцов и таблиц

ALTER TABLE используется для добавления, удаления или изменения столбцов в существующей таблице, а также для добавления и удаления различных ограничений в существующей таблице.

Рассмотрим следующую таблицу People:

ID	FirstName	LastName	City
1	John	Smith	New York
2	David	Williams	Los Angeles
3	Chloe	Anderson	Chicago

Следующий код ДОБАВЛЯЕТ новый столбец с именем DateOfBirth :

```
ALTER TABLE People
ADD DateOfBirth date;
```

ID	FirstName	LastName	City	DateOfBirth
1	John	Smith	New York	*NULL*
2	David	Williams	Los Angeles	*NULL*
3	Chloe	Anderson	Chicago	*NULL*

Следующий код УДАЛЯЕТ столбец с именем DateOfBirth :

```
ALTER TABLE People
DROP COLUMN DateOfBirth;
```

Чтобы УДАЛИТЬ ВСЮ ТАБЛИЦУ, используйте команду DROP TABLE :

```
DROP TABLE People;
```

УДАЛИТЬ всю базу данных:

```
DROP DATABASE People;
```

Команда ALTER TABLE также используется для ПЕРЕИМЕНОВАНИЯ СТОЛБЦОВ.
Этот запрос переименует столбец с именем FirstName на Name.

```
ALTER TABLE People
CHANGE FirstName Name
varchar (100);
```

Вы можете ПЕРЕИМЕНОВАТЬ ВСЮ ТАБЛИЦУ с помощью команды RENAME :

```
RENAME TABLE People TO Users;
```

SHOW - Получение информации по базам данных, таблицам, столбцам

Отображение всех БД:
```
SHOW DATABASES;
```

Отображение всех таблиц в текущей БД:
```
SHOW TABLES;
```

Отображает информацию о столбцах в заданной таблице: `SHOW COLUMNS FROM `customers`;`

INSERT - Добавление данных в таблицу

Добавление данных в таблицу customers с 4 столбцами: «id», «firstname», «email» и «reg_date»:
```
INSERT INTO `customers` VALUES
(NULL, 'Alex', 'alex@gmail.com', NULL);
```
NULL указано для полей в которые MySQL сам добавит значения, так как поле «id» имеет AUTO_INCREMENT, а поле «reg_date» имеет тип данных TIMESTAMP, значение которого по умолчанию равно NOW().

Добавление сразу нескольких строк:
```
INSERT INTO `customers` VALUES
(NULL, 'Alex', 'alex@gmail.com', NULL),
(NULL, 'Hill', 'hill@gmail.com', NULL);
```

Добавление данных в таблицу customers в поля «firstname» и «email»:
```
INSERT INTO `customers` (`firstname`, `email`)
VALUES ('Alex', 'alex@gmail.com');
```
Так как поле «id» имеет AUTO_INCREMENT, а поле «reg_date» – TIMESTAMP, нет необходимости их указывать, MySQL автоматически добавит значение для них.

Добавление данных в таблицу customers в поля «firstname» и «email» можно выполнить и так:
```
INSERT INTO `customers`
SET `firstname`='Alex', `email`='alex@gmail.com';
```

UPDATE - Обновление записей

```
UPDATE `customers`
SET `firstname`='Joe'
WHERE `id`=1;
```
Если условие WHERE не указано – будут обновлены все записи firstname во всей таблице.

DELETE - Удаление записей

```
DELETE FROM `customers`
WHERE `id`=1;
```
Если условие WHERE не указано – будут удалены все записи во всей таблице.

SELECT - Выбор записей

Синтаксис:
```
SELECT поле1, поле2, ...
FROM имяТаблицы
[ WHERE условие
  GROUP BY поле HAVING условие
  ORDER BY поле DESC
  LIMIT начало, число записей ]
```

Выбрать все поля в таблице «customers»:
```
SELECT * FROM `customers`;
```

Выбрать все поля в таблице «customers» с id = 2:
```
SELECT * FROM `customers`
WHERE `id` = 2;
```

Выбрать поля «firstname» и «email» из таблицы «customers»:
```
SELECT `firstname`, `email`
FROM `customers`;
```

Можно указать имя таблицы перед именем столбца, разделяя их точкой. След записи эквивалентны:
`SELECT `firstname` FROM `customers` === SELECT ` customers.firstname` FROM `customers`;`
Такая форма написания полезна при работе со множеством таблиц, которые имеют одинаковые названия столбцов.

DISTINCT

SELECT DISTINCT – выбирает уникальные записи, отбрасывая повторения записей.

Singer	Album	Year	Sale
The Prodigy	Invaders Must Die	2008	1200000
Drowning Pool	Sinner	2001	400000
Massive Attack	Mezzanine	1998	2300000
The Prodigy	Fat of the Land	1997	600000
The Prodigy	Music For The Jilted Generation	1994	1500000
Massive Attack	100th Window	2003	1200000
Drowning Pool	Full Circle	2007	800000
Massive Attack	Danny The Dog	2004	1900000
Drowning Pool	Resilience	2013	500000

Пример 1. Вывести, какие исполнители имеются в таблице:

```
SELECT DISTINCT Singer
FROM Artists;
```

Результат:
Singer
The Prodigy
Drowning Pool
Massive Attack

Пример 2. Вывести, количество уникальных исполнителей в таблице:

```
SELECT COUNT(DISTINCT Singer)
FROM Artists;
```

Результат:
3

ORDER BY

ORDER BY – используется для сортировки возвращаемой информации в возрастающем порядке (по алфавиту, если строка).

Выбрать поля «firstname» и «email» в таблице «customers» и отсортировать их по полю «email»:

```
SELECT `firstname`, `email`
FROM `customers`
ORDER BY `email`;
```

ORDER BY может сортировать информацию по нескольким столбцам:

```
SELECT `firstname`, `email`
FROM `customers`
ORDER BY `firstname`, `email`;
```

Таблица до сортировки:

firstname	email
b	4
c	3
a	1
b	2

Таблица после сортировки:

firstname	email
a	1
b	2
b	4
c	3

Так как у нас 2 значения b – они будут отсортированы по столбцу email в возрастающем порядке.
ORDER BY начинает сортировку в том же порядке, в котором находятся столбцы.
Сначала будет отсортирован первый столбец firstname, потом второй email и т. д.

LIMIT

LIMIT - позволяет вывести указанное число строк из таблицы, имеет 2 значения – начало, число строк.

Выбрать 2 записи «firstname» и «email» в таблице «customers», выбор начать с 4 записи (первая имеет номер 0, а вторая - номер 1):

```
SELECT `firstname`, `email` FROM `customers` LIMIT 3, 2;
```

Выбрать первые 5 записей:

```
SELECT `firstname`, `email` FROM `customers` LIMIT 5;
```

WHERE

WHERE используется для выполнения только тех записей, которые соответствуют условию.
Для фильтрации информации в WHERE используются условные и логические операторы.
WHERE может включать в себя предикаты AND, OR, NOT, LIKE, BETWEEN, IN, ключевое слово NULL, операторы сравнения и равенства =, !=, >, <, >=, <=.

AND – true, если оба выражения true
OR – true, если одно из выражений true
IN – true, если операнд равен одному из выражений в списке
NOT – true, если выражение не true

При комбинации этих операторов важно использовать скобки:
```
SELECT * FROM customers WHERE city = "New York" AND (age = 30 OR Age = 35);
```

BETWEEN

BETWEEN задает диапазон, в котором будет осуществляться проверка условия.
Синтаксис:
```
[NOT] BETWEEN begin AND end
```
begin — начальное значение диапазона;
end — конечное значение диапазона;
```
SELECT * FROM Universities
WHERE Students BETWEEN 100 AND 300
```

IN

IN используется, когда вы хотите сравнить столбец больше чем с одним выражением.
Например, нужно выбрать всех клиентов из New York, Los Angeles и Chicago. С условием OR запрос будет выглядеть так:
```
SELECT * FROM customers
WHERE City = 'New York'
OR City = 'Los Angeles'
OR City = 'Chicago';
```
Можно получить такой же результат с одним условием IN:
```
SELECT * FROM customers
WHERE City IN ('New York', 'Los Angeles', 'Chicago');
```

NOT IN исключает значения из набора. Например, нужно исключить всех клиентов из New York, Los Angeles и Chicago:
```
SELECT * FROM customers
WHERE City NOT IN ('New York', 'Los Angeles', 'Chicago');
```

LIKE

LIKE проверяет соответствие шаблону pattern:
```
SELECT column_name
FROM table_name
WHERE column_name LIKE pattern;
```

pattern — шаблон, по которому будет происходить проверка выражения. Шаблон может включать в себя следующие спец. символы:

%	Строка любой длины
_	Любой одиночный символ
[]	Диапазон или последовательность символов
[^]	Исключающий диапазон или последовательность символов

Выбрать работников, чьи FirstName начинаются с буквы A:
```
SELECT * FROM employees
WHERE FirstName LIKE 'A%';
```

Выбрать работников, чьи FirstName заканчиваются буквой s:
```
SELECT * FROM employees
WHERE FirstName LIKE '%s';
```

Арифметические операторы + - * /

Выполняют арифметические операции с числовыми операндами.
Добавим 500 к зарплате каждого работника:
```
SELECT ID, FirstName, LastName, Salary+500 AS Salary
FROM employees;
```

Функция CONCAT

Используется для конкатенации текстовых значений и для возврата текстовой строки.
Например, соединим FirstName и City, разделив их запятой:
```
SELECT CONCAT(FirstName, ', ' , City)
FROM customers;
```
Будет получен такой результат:
CONCAT(FirstName, ', ' , City)
John, New York
Dave, Chicago

Ключевое слово AS

Конкатенация образует новый столбец CONCAT.
Изменить имя столбцу можно с помощью AS:

```
SELECT CONCAT(FirstName,', ', City) AS new_column
FROM customers;
```

При запуске запроса название столбца будет изменено:
new_column
John, New York
Dave, Chicago

Обратите внимание, что оператор AS может использоваться без функции конкатенации для переименования существующих столбцов; например, если вам нужно представить результаты вашему боссу или что-то еще. Например, у вас есть клиенты таблиц с столбцами FirstName, LastName, PhoneNo, и вы хотите иметь хороший заголовок для отчета или что-то еще, вы можете написать запрос типа:

```
SELECT FirstName AS "First name",
LastName AS "Last name",
PhoneNo AS "Phone number"
FROM customers;
```

и в результате получится красивое имя столбца.
Также обратите внимание, что в моем примере я использовал несколько слов, разделенных пробелами в именах. Для этого вам нужно поместить все имя в кавычки (а не апострофы, как мы используем для простых текстовых строк в CONCAT) после оператора AS.

Ф-ция COUNT

COUNT() – возвращает количество непустых (не имеющих значение NULL) строк в указанном поле:

```
SELECT COUNT(`id_Orders`) FROM `Orders`;
```

Ф-ция MIN() / MAX()

MIN() / MAX() – возвращают минимальное / максимальное значение в указанном поле.
Узнаем минимальную ЗП среди работников:

```
SELECT MIN(Salary) AS Salary FROM employees;
```

Ф-ция UPPER / LOWER

Конвертируют строку в верхний/нижний регистр. Если в строке есть символы «не буквы» - то ф-ция не будет влиять на них.

```
SELECT FirstName, UPPER(LastName) AS LastName
FROM employees;
```

Ф-ция SQRT и AVG

SQRT возвращает квадратный корень заданного значения в аргументе.
AVG возвращает среднее значение числового столбца.

```
SELECT Salary, SQRT(Salary)
FROM employees;
```

Ф-ция SUM

SUM - подсчет суммы значений столбцов.
Например, подсчет суммы всех зарплат:

```
SELECT SUM(Salary) FROM employees;
```

Подзапросы

Подзапрос – это запрос внутри другого запроса.
Например, нам нужны работники, чьи ЗП выше средней. Сперва подсчитаем среднюю ЗП:

```
SELECT AVG(Salary) FROM employees;
```

Теперь, когда мы знаем среднюю ЗП, можно использовать выражение WHERE для отображения списка ЗП, которые больше этого числа:

```
SELECT FirstName, Salary FROM employees
WHERE  Salary > 3100
ORDER BY Salary DESC;
```

DESC сортирует результаты в порядке убывания, ASC в порядке возрастания.

Подзапрос выведет такой же результат намного проще:

```
SELECT FirstName, Salary FROM employees
WHERE  Salary > (SELECT AVG(Salary) FROM employees)
ORDER BY Salary DESC;
```

Соединение таблиц

ID	Name	City	Age
1	John	New York	35
2	David	Los Angeles	23
3	Chloe	Chicago	27
4	Emily	Houston	34
5	James	Philadelphia	31

ıe **orders** table stores information about individual orders with their corresponding amount:

ID	Name	Customer_ID	Amount
1	Book	3	5000
2	Box	5	3000
3	Toy	2	4500
4	Flowers	4	1800
5	Cake	1	6700

Для того, чтоб объединить таблицы, укажите их в списке через запятую в FROM:

```
SELECT customers.ID, customers.Name, orders.Name, orders.Amount
FROM customers, orders
WHERE customers.ID=orders.Customer_ID
ORDER BY customers.ID;
```

В возвращенной информации отображены заказы клиента и их количество:

ID	Name	Name	Amount
1	John	Cake	6700
2	David	Toy	4500
3	Chloe	Book	5000
4	Emily	Flowers	1800
5	James	Box	3000

Установка сокращенного «прозвища» для таблицы

Можно сократить запрос с помощью установки таблице «прозвища». Запрос, написанный выше, можно написать так:

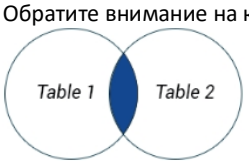
```
SELECT ct.ID, ct.Name, ord.Name, ord.Amount
FROM customers AS ct, orders AS ord
WHERE ct.ID=ord.Customer_ID
ORDER BY ct.ID;
```

Типы объединений INNER JOIN, LEFT JOIN, RIGHT JOIN

- INNER JOIN (внутреннее объединение) – возвращает строки, если они совпадают в таблицах. Синтаксис

```
SELECT column_name(s)
FROM table1 INNER JOIN table2
ON table1.column_name=table2.column_name;
```

Обратите внимание на ключевое слово ON, указывающее на условие внутреннего объединения.



Изображение демонстрирует, как работает INNER JOIN.
Возвращаются только совпадающие записи.

- LEFT JOIN возвращает все строки из левой таблицы, даже если нет соответствий в правой. Синтаксис:

```
SELECT table1.column1, table2.column2...
FROM table1 LEFT OUTER JOIN table2
ON table1.column_name = table2.column_name;
```

Ключевое слово OUTER – опционально и может быть опущено.
Изображение демонстрирует, как работает LEFT JOIN:

Рассмотрим таблицы:

customers:

ID	Name	LastName	City
1	John	Smith	New York
2	David	Williams	Los Angeles
3	Chloe	Anderson	Chicago
4	Emily	Adams	Houston
5	James	Roberts	Philadelphia
6	Andrew	Thomas	New York
7	Daniel	Harris	New York

items:

ID	Name	Cost	Seller_id
39	Book	5.9	1
24	Box	2.99	1
72	Toy	23.7	2
36	Flowers	50.75	2
18	T-Shirt	22.5	3
16	Notebook	150.22	4
74	Perfume	90.9	6

Данный запрос выведет всех клиентов и все вещи, которые у них есть:

```
SELECT customers.Name, items.Name
FROM customers LEFT OUTER JOIN items
ON customers.ID=items.Seller_id;
```

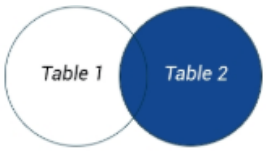
Result:

Name	Name
John	Book
John	Box
David	Toy
David	Flowers
Chloe	T-Shirt
Emily	Notebook
Andrew	Perfume
James	*NULL*
Daniel	*NULL*

В результате имеются все строки из левой таблицы и все данные из правой таблицы.
Если совпадение не найдено, то будет возвращено NULL.

- RIGHT JOIN – возвращает все строки из правой таблицы, даже если нет соответствий в левой таблице. Синтаксис:

```
SELECT table1.column1, table2.column2...
FROM table1 RIGHT OUTER JOIN table2
ON table1.column_name = table2.column_name;
```



Ключевое слово OUTER – опционально и может быть опущено.

Рассмотрим тот же пример, но для RIGHT JOIN:

```
SELECT customers.Name, items.Name FROM customers
RIGHT JOIN items ON customers.ID=items.Seller_id;
```

Result:

Name	Name
John	Book
John	Box
David	Toy
David	Flowers
Chloe	T-Shirt
Emily	Notebook
Andrew	Perfume

Объединение RIGHT JOIN возвращает все строки из правой таблицы items даже при отсутствии соответствий в левой таблице customers.

UNION / UNION ALL

UNION используется для объединения двух и более запросов оператора SELECT и удаляет все дубликаты.
UNION ALL используется для объединения двух и более запросов оператора SELECT, но не удаляет дублирующиеся строки.
UNION ALL быстрее, чем UNION , поскольку он не выполняет операцию удаления дубликатов.
Важно отметить, что каждый из операторов SELECT должен иметь в своем запросе одинаковое количество столбцов и одинаковые типы данных, кроме того, столбцы в каждой инструкции SELECT должны быть в том же порядке.

Пример, есть 2 таблицы:

ID	FirstName	LastName	City
1	John	Smith	New York
2	David	Williams	Los Angeles

ID	FirstName	LastName	City
1	James	Roberts	Philadelphia
2	David	Williams	Los Angeles

Запрос UNION:

```
SELECT ID, FirstName, LastName, City FROM First
UNION
SELECT ID, FirstName, LastName, City FROM Second;
```

Полученная таблица будет выглядеть следующим образом (дубликаты были удалены):

ID	FirstName	LastName	City
1	John	Smith	New York
2	David	Williams	Los Angeles
1	James	Roberts	Philadelphia

СОВЕТ. Если ваши столбцы не совпадают точно по всем запросам, вы можете использовать значение NULL (или любое другое), например:

```
SELECT FirstName, LastName, Company FROM businessContacts
UNION
SELECT FirstName, LastName, NULL FROM otherContacts;
```

Запрос UNION ALL для тех же таблиц:

```
SELECT ID, FirstName, LastName, City FROM First
UNION ALL
SELECT ID, FirstName, LastName, City FROM Second;
```

Результат содержит дубликаты строк:

ID	FirstName	LastName	City
1	John	Smith	New York
2	David	Williams	Los Angeles
1	James	Roberts	Philadelphia
2	David	Williams	Los Angeles

Виртуальная таблица VIEW

В виртуальной таблице содержатся строки и столбцы, как и в реальной таблице. Поля представляют собой поля из одной или нескольких реальных таблиц в базе данных.
Виртуальные таблицы позволяют нам:
- Структурировать данные таким способом, который пользователи считают интуитивно понятным.
- Ограничивать доступ к данным таким образом, чтобы пользователь мог видеть и изменять именно то, что ему нужно, и не более того.
- Суммировать данные из разных таблиц и использовать их для создания отчетов.

Синтаксис:

```
CREATE VIEW  view_name AS
SELECT column_name (s)
FROM table_name
WHERE условие;
```

Рассмотрим таблицу Employees , которая содержит следующие записи:

ID	FirstName	LastName	Age	Salary
1	Emily	**Adams**	**34**	**5000**
2	Chloe	**Anderson**	**27**	**10000**
3	Daniel	**Harris**	**30**	**6500**
4	James	**Roberts**	**31**	**5500**
5	John	**Smith**	**35**	**4500**
6	Andrew	**Thomas**	**45**	**6000**
7	David	**Williams**	**23**	**3000**

Давайте СОЗДАДИМ ПРЕДСТАВЛЕНИЕ, отображающее FirstName и Salary каждого сотрудника:

```
CREATE VIEW List AS
SELECT FirstName, Salary
FROM Employees;
```

Теперь вы можете запросить представление «List», как если бы вы запрашивали фактическую таблицу:

```
SELECT * FROM List;
```

FirstName	Salary
Emily	5000
Chloe	10000
Daniel	6500
James	5500
John	4500
Andrew	6000
David	3000

Представление всегда отображает обновленные данные!

Вы можете ОБНОВИТЬ ПРЕДСТАВЛЕНИЕ, используя следующий синтаксис:

```
CREATE OR REPLACE VIEW view_name AS
SELECT column_name (s)
FROM table_name
WHERE условие;
```

В приведенном ниже примере обновляется наше представление List для выбора LastName:

```
CREATE OR REPLACE VIEW List AS
SELECT FirstName, LastName, Salary
FROM Employees;
```

Результат:

FirstName	LastName	Salary
Emily	Adams	5000
Chloe	Anderson	10000
Daniel	Harris	6500
James	Roberts	5500
John	Smith	4500
Andrew	Thomas	6000
David	Williams	3000

УДАЛИТЬ представление с помощью команды DROP VIEW :

```
DROP VIEW List;
```

====

PK – primary key – первичный ключ
FK – foreign key – внешний ключ
AK – alternate key

https://ru.wikipedia.org/wiki/%D0%92%D0%BD%D0%B5%D1%88%D0%BD%D0%B8%D0%B9_%D0%BA%D0%BB%D1%8E%D1%87