

УВАЖАЕМЫЕ СТУДЕНТЫ! Изучите теоретические сведения к лабораторной работе, выполните задания лабораторной работы.

Результаты работы, отчет, предоставить преподавателю на e-mail: xvsviv@rambler.ru

Требования к отчету:

Отчет предоставляется преподавателю в электронном варианте и должен содержать:

- название работы, постановку цели, вывод;
- скриншоты поэтапного выполнения лабораторной работы.

При возникновении вопросов по приведенному материалу обращаться по номеру телефона: 0721389311 (WhatsApp).

ВНИМАНИЕ!!! При отправке работы, не забывайте указывать ФИО студента, наименование дисциплины, дата проведения занятия (по расписанию).

Лабораторная работа. «Организация запросов на выборку данных. Структура команды Select. Сортировка результатов запроса. Изучение итоговых функций и средств группировки данных»

Цель: изучить структуру команды Select, научиться выполнять сортировку, группировку результатов запроса.

Ограничения целостности столбцов и таблиц БД можно изменять, а также запрещать, разрешать и удалять. Это дает разработчику возможность создавать, модифицировать и удалять бизнес-правила, ограничивающие данные.

[Существует ряд условий создания ограничений:](#)

Первичные ключи: в столбцах не могут содержаться NULL-значения, и все значения должны быть уникальны.

Внешние ключи: в тех столбцах других таблиц, на которые производятся ссылки, должны содержаться значения, соответствующие всем значениям

ссылающихся столбцов, либо значения этих последних должны быть NULL-значениями.

Ограничения UNIQUE: все значения столбцов должны быть уникальными или NULL-значениями.

Ограничения CHECK: новое ограничение будет применяться только по отношению к данным, добавляемым или модифицируемым после его создания.

NOT NULL: NULL-значения в столбцах запрещены.

Ограничения можно разрешать и запрещать. Разрешенное ограничение выполняет свои функции, реализуя бизнес-правила по отношению к вводимым в таблицу данным, а запрещенное ограничение переводится в разряд недействующих, как если бы оно было удалено, и его правила не реализуются.

Задание 1.

Создать базу данных My_bd2.

CREATE DATABASE My_bd2

В этой базе данных создаём таблицу:

Часто для наименования поля идентификатора используется слово ID.

Теперь создадим нашу таблицу:

CREATE TABLE Employees(

ID int,

Name varchar(30),

Birthday date,

Email varchar(30),

Position varchar(30),

Department varchar(30)

)

Для того, чтобы задать обязательные для заполнения столбцы, можно использовать опцию NOT NULL.

Для уже существующей таблицы поля можно переопределить при помощи следующих команд:

```
-- обновление поля ID и поля Name
ALTER TABLE Employees ALTER COLUMN ID int NOT NULL;
ALTER TABLE Employees ALTER COLUMN Name varchar(30) NOT
NULL;
```

При выполнении этих команд SQL выдает ошибку, так опцию NOT NULL в данной версии SQL нужно использовать непосредственно при создании новой таблицы/

Удалим таблицу и создадим ее заново с ограничением NOT NULL:

```
DROP TABLE Employees
```

Создаем таблицу:

```
CREATE TABLE Employees(
  ID int NOT NULL,
  Name varchar(30) NOT NULL,
  Birthday date,
  Email varchar(30),
  Position varchar(30),
  Department varchar(30)
)
```

Задание 2

При создании таблицы желательно, чтобы она имела уникальный столбец или же совокупность столбцов, которая уникальна для каждой ее строки – по данному уникальному значению можно однозначно идентифицировать запись. Такое значение называется первичным ключом

таблицы. Для нашей таблицы Employees таким уникальным значением может быть столбец ID (который содержит «Табельный номер сотрудника» — пускай в нашем случае данное значение уникально для каждого сотрудника и не может повторяться).

Создать первичный ключ к уже существующей таблице можно при помощи команды:

```
ALTER TABLE Employees ADD CONSTRAINT PK_Employees  
PRIMARY KEY(ID)
```

где «PK_Employees» это имя ограничения, отвечающего за первичный ключ. Обычно для наименования первичного ключа используется префикс «PK_» после которого идет имя таблицы.

Задание 3

После создания зальем в таблицу данные:

```
INSERT Employees(ID,Position,Department,Name) VALUES  
(1000,N'Директор',N'Администрация',N'Иванов И.И.'),  
(1001,N'Программист',N'ИТ',N'Петров П.П.'),  
(1002,N'Бухгалтер',N'Бухгалтерия',N'Сидоров С.С.'),  
(1003,N'Старший программист',N'ИТ',N'Андреев А.А.')
```

Задание 4

В MS SQL Server существует два вида временных таблиц: локальные и глобальные. Локальные временные таблицы видны только их создателям до завершения сеанса соединения с экземпляром SQL Server, как только они

впервые созданы. Локальные временные таблицы автоматически удаляются после отключения пользователя от экземпляра SQL Server. Глобальные временные таблицы видны всем пользователям в течение любых сеансов соединения после создания этих таблиц и удаляются, когда все пользователи, ссылающиеся на эти таблицы, отключаются от экземпляра SQL Server.

Временные таблицы создаются в системной базе tempdb, т.е. создавая их мы не засоряем основную базу, в остальном же временные таблицы полностью идентичны обычным таблицам, их так же можно удалить при помощи команды DROP TABLE. Чаще используются локальные временные таблицы.

Для создания временной таблицы можно использовать команду CREATE TABLE:

```
CREATE TEMPORARY TABLE Temp (  
  ID int,  
  Name varchar(30)  
)
```

Так же временную таблицу (как собственно и обычную таблицу) можно создать и сразу заполнить данными возвращаемые запросом используя синтаксис SELECT ... INTO:

```
SELECT ID,Name  
INTO Temp  
FROM Employees
```

На заметку

В разных СУБД реализация временных таблиц может отличаться. Например, в СУБД ORACLE и Firebird структура временных таблиц должна быть определена заранее командой CREATE GLOBAL TEMPORARY TABLE

с указанием специфики хранения в ней данных, дальше уже пользователь видит ее среди основных таблиц и работает с ней как с обычной таблицей.

Задание 5

Для нормализации базы данных создадим по очереди 2 таблицы справочники «Должности» и «Отделы», первую назовем Positions, а вторую соответственно Departments:

```
CREATE TABLE Positions(  
  ID int NOT NULL AUTO_INCREMENT PRIMARY KEY,  
  Name nvarchar(30) NOT NULL  
)  
  
CREATE TABLE Departments(  
  ID int NOT NULL AUTO_INCREMENT PRIMARY KEY,  
  Name nvarchar(30) NOT NULL  
)
```

Заметим, что здесь мы использовали новую опцию AUTO_INCREMENT, которая говорит о том, что данные в столбце ID будут нумероваться автоматически, начиная с 1, с шагом 1, т.е. при добавлении новых записей им последовательно будут присваиваться значения 1, 2, 3, и т.д. Такие поля обычно называют автоинкрементными. В таблице может быть определено только одно поле со свойством AUTO_INCREMENT и обычно, но необязательно, такое поле является первичным ключом для данной таблицы.

Проверьте созданные таблицы просмотрев их структуру.

Задание 6

Заполним эти таблицы автоматически, на основании текущих данных записанных в полях Position и Department таблицы Employees:

-- заполняем поле Name таблицы Positions, уникальными значениями из поля Position таблицы Employees

```
INSERT INTO Positions(Name)
```

```
SELECT DISTINCT Position
```

```
FROM Employees
```

```
WHERE Position IS NOT NULL -- отбрасываем записи у которых  
позиция не указана
```

То же самое сделаем для таблицы Departments:

```
INSERT INTO Departments(Name)
```

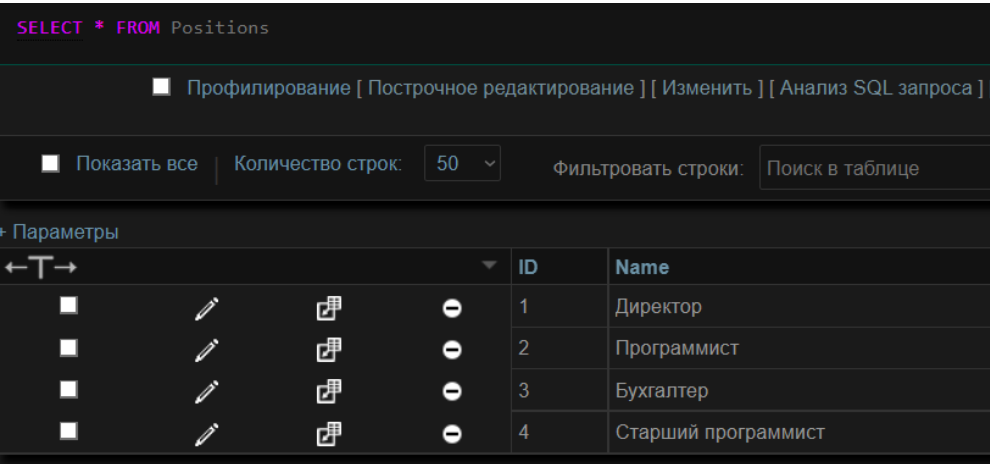
```
SELECT DISTINCT Department
```

```
FROM Employees
```

```
WHERE Department IS NOT NULL
```

Если теперь мы откроем таблицы Positions и Departments, то увидим пронумерованный набор значений по полю ID:

```
SELECT * FROM Positions
```



SELECT * FROM Positions

Профилирование [Построчное редактирование] [Изменить] [Анализ SQL запроса]

Показать все | Количество строк: 50 | фильтровать строки: Поиск в таблице

Параметры

	ID	Name
<input type="checkbox"/>	1	Директор
<input type="checkbox"/>	2	Программист
<input type="checkbox"/>	3	Бухгалтер
<input type="checkbox"/>	4	Старший программист

Самостоятельно просмотрите содержание таблицы Departments.

Задание 7

Данные таблицы теперь и будут играть роль справочников для задания должностей и отделов. Теперь мы будем ссылаться на идентификаторы должностей и отделов. В первую очередь создадим новые поля в таблице Employees для хранения данных идентификаторов. Тип ссылочных полей должен быть каким же, как и в справочниках, в данном случае это int. Добавить в таблицу сразу несколько полей можно одной командой, перечислив поля через запятую:

```
ALTER TABLE Employees ADD(PositionID int, DepartmentID int)
```

Теперь пропишем ссылки (ссылочные ограничения — FOREIGN KEY) для этих полей, для того чтобы пользователь не имел возможности записать в данные поля, значения, отсутствующие среди значений ID находящихся в справочниках.

```
ALTER TABLE Employees ADD CONSTRAINT  
FK_Employees_PositionID  
FOREIGN KEY(PositionID) REFERENCES Positions(ID)
```

И то же самое сделаем для второго поля:

```
ALTER TABLE Employees ADD CONSTRAINT  
FK_Employees_DepartmentID  
FOREIGN KEY(DepartmentID) REFERENCES Departments(ID)
```

Теперь пользователь в данные поля сможет занести только значения ID из соответствующего справочника. Соответственно, чтобы использовать

новый отдел или должность, он первым делом должен будет добавить новую запись в соответствующий справочник. Т.к. должности и отделы теперь хранятся в справочниках в одном единственном экземпляре, то чтобы изменить название, достаточно изменить его только в справочнике.

Имя ссылочного ограничения, обычно является составным, оно состоит из префикса «FK_», затем идет имя таблицы и после знака подчеркивания идет имя поля, которое ссылается на идентификатор таблицы-справочника.

Идентификатор (ID) обычно является внутренним значением, которое используется только для связей и какое значение там хранится, в большинстве случаев абсолютно безразлично, поэтому не нужно пытаться избавиться от дырок в последовательности чисел, которые возникают по ходу работы с таблицей, например, после удаления записей из справочника.

Задание 8

Обновим поля PositionID и DepartmentID значениями ID из справочников. Воспользуемся для этой цели DML командой UPDATE:

```
UPDATE Employees
SET
    PositionID=(SELECT ID FROM Positions WHERE Name= Position),
    DepartmentID=(SELECT ID FROM Departments WHERE
Name=Department)
```

Посмотрим, что получилось, выполнив запрос:

```
SELECT * FROM Employees
```

Всё, поля PositionID и DepartmentID заполнены соответствующие должностям и отделам идентификаторами надобности в полях Position и Department в таблице Employees теперь нет, можно удалить эти поля:

`ALTER TABLE employees DROP COLUMN positions, DROP COLUMN Department`

Теперь таблица у нас приобрела следующий вид:

```
SELECT * FROM Employees
```

#	Имя	Тип	Сравнение	Атрибуты	Null	По умолчанию
1	ID 🔑	int(11)			Нет	Нет
2	Name	varchar(30)	utf8mb4_unicode_ci		Нет	Нет
3	Birthday	date			Да	NULL
4	Email	varchar(30)	utf8mb4_unicode_ci		Да	NULL
5	PositionID 🔑	int(11)			Да	NULL
6	DepartmentID 🔑	int(11)			Да	NULL

Т.е. мы в итоге избавились от хранения избыточной информации. Теперь, по номерам должности и отдела можем однозначно определить их названия, используя значения в таблицах-справочниках:

Задание 9

В итоге избавились от хранения избыточной информации. Теперь, по номерам должности и отдела можем однозначно определить их названия, используя значения в таблицах-справочниках:

```
SELECT (ID, Name, PositionName, DepartmentName)
FROM Employees
LEFT JOIN Departments ON ID=DepartmentID
LEFT JOIN Positions p ON ID=PositionID
```

Контрольные вопросы:

1. Запишите инструкцию создания базы данных, создания таблицы, добавления или удаления столбца таблицы.
2. Какие еще действия с таблицей относятся к модификации таблицы. Запишите их инструкции.