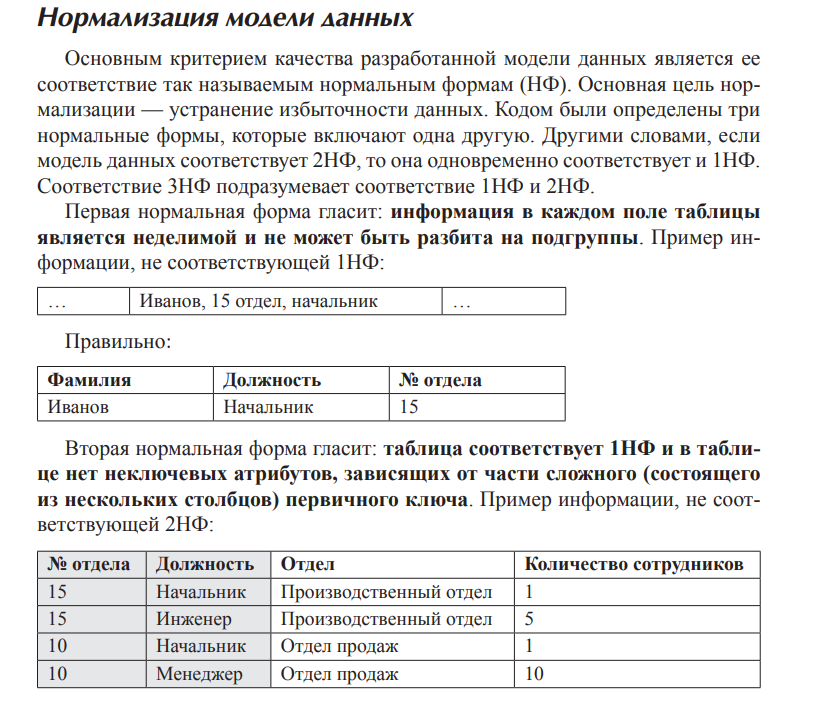
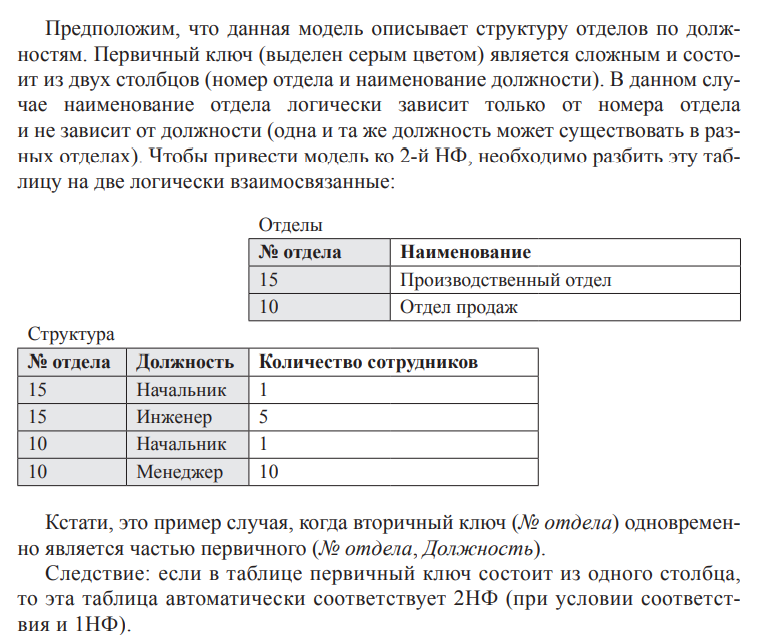
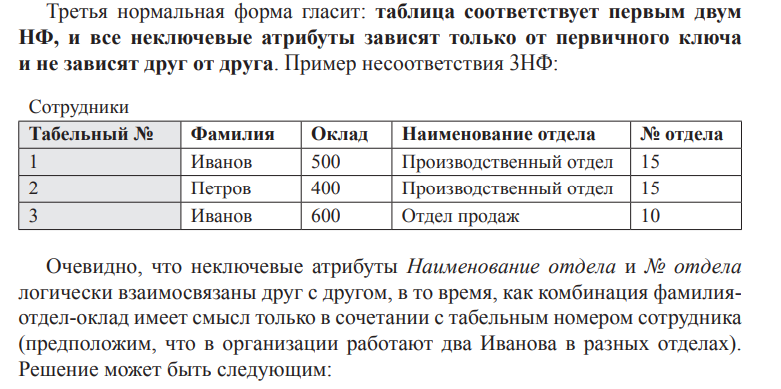
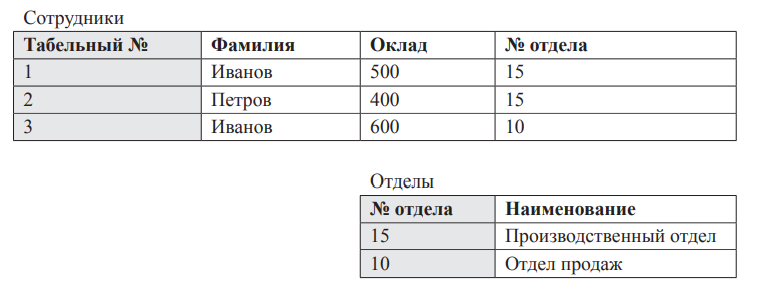
**SQL**

1. Что такое нормализация?







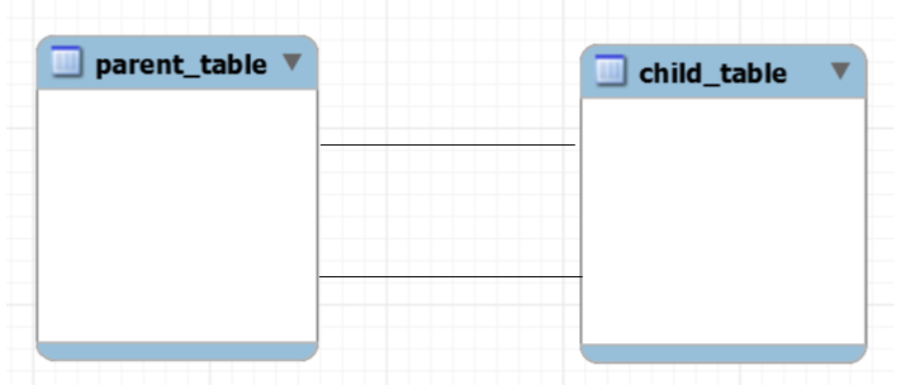


1. Какие есть типы связей в базе данных. Привести пример.

Существует **три вида связей между таблицами баз данных**:  
• связь один к одному;  
• связь один ко многим;  
• связь многие ко многим.

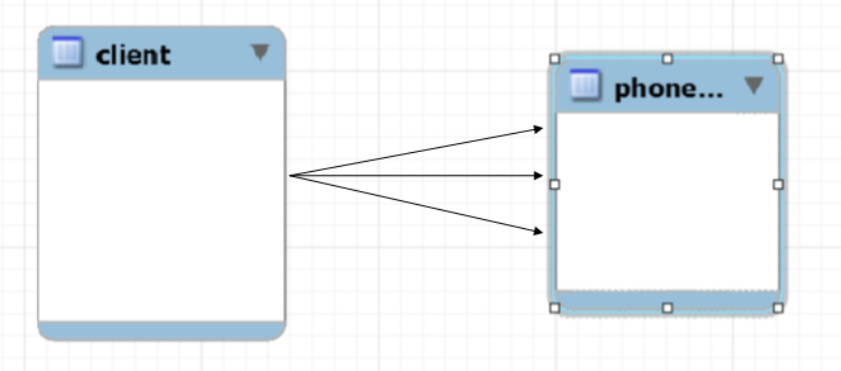
**Связь один к одному** – самая редко встречаемая связь между таблицами. В 97 случаях из 100, если вы видите такую связь, вам необходимо объединить две таблицы в одну.

Таблицы будут **связаны один к одному** тогда, когда одному объекту таблицы А соответствует один объект таблицы Б, и одному объекту таблицы Б соответствует один объект таблицы А. Как я уже говорил: если вы видите, что **связь один к одному** – смело объединяйте таблицы в одну, за исключением тех случаев, когда происходит модернизация базы данных.  
Например, у нас была таблица, в которой хранились данные о сотрудниках компании. Но произошли какие-то изменения в бизнес-процессе и появилась необходимость создать таблицы с теми же самыми сотрудниками, но не для всей компании, а разбив их по отделам. Таблицы отделов будут дочерними по отношению к таблице, в которой хранятся данные обо всех сотрудниках компании, и связаны такие таблицы будут связью один к одному.



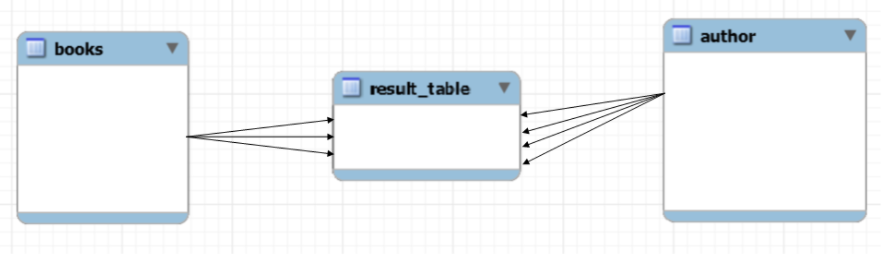
**Связь один ко многим в реляционных базах данных** реализуется тогда, когда объекту А может принадлежать или же соответствовать несколько объектов Б, но объекту Б может соответствовать только один объект А.

У нас есть таблица, в которой содержатся данные о клиентах и у нас есть таблица, в которой хранятся их телефоны. Мы можем смело утверждать, что у одного клиента может быть несколько телефонов, но в тоже время мы можем быть уверены в том, что один конкретный номер может быть только у одного клиента. Это типичный **пример связи один ко многим**.

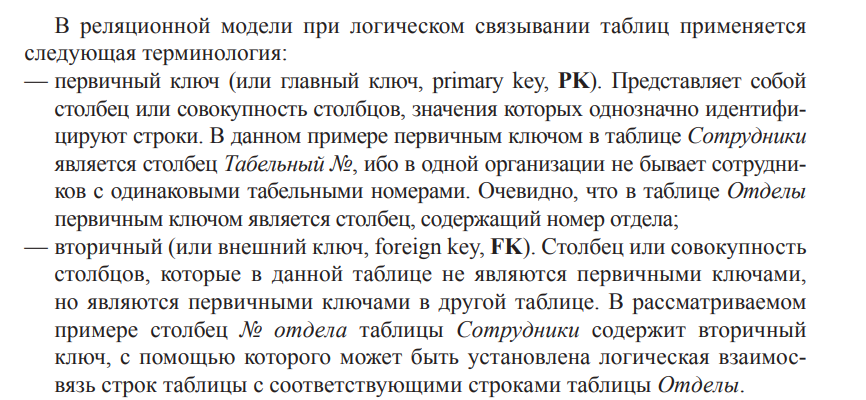


**Связь многие ко многим** реализуется в том случае, когда нескольким объектам из таблицы А может соответствовать несколько объектов из таблицы Б, и в тоже время нескольким объектам из таблицы Б соответствует несколько объектов из таблицы А. Рассмотрим простой пример.

У нас есть таблица с книгами и есть таблица с авторами. Приведу два верных утверждения. Первое: одну книгу может написать несколько авторов. Второе: автор может написать несколько книг. Здесь мы наблюдаем типичную ситуацию, когда **связь между таблицами многие ко многим**. Такая связь (**связь многие ко многим**) реализуется путем добавления третьей таблицы.



1. Что такое primary key (первичный ключ)?
2. Что такое foreign key (внешний ключ)?



1. Что такое индексы в базе данных? Для чего их используют? Чем они хороши и чем плохи?

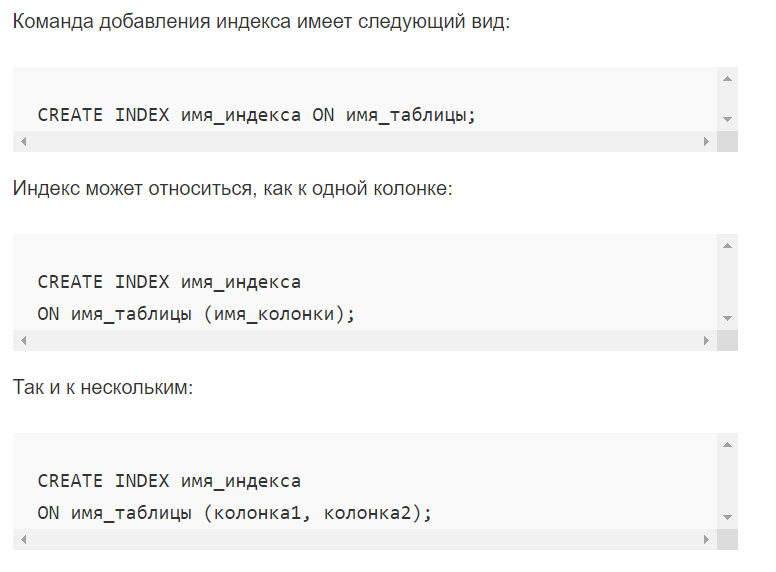
Индексы – это специальные таблицы, которые могут быть использованы поисковым двигателем БД для ускорения получения данных. Необходимо просто добавить указатель индекса в таблицу. Индекс в БД крайне схож с индексом в конце книги.

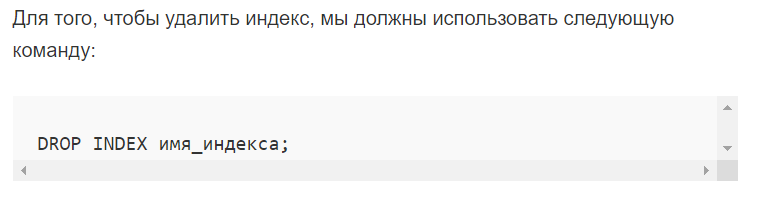
Пример:

Допустим, мы хотим иметь ссылку на все страницы книги, которые касаются определённой темы (например, Наследование в книге по программированию на языке Java). Для этого, мы в первую очередь ссылаемся на индекс, который указан в конце книге и переходим на любую из страниц, которая относится к необходимой теме.

Плюсы: индекс помогает ускорить запросы на получение данных (SELECT [WHERE]).

Минусы: индекс замедляет процесс добавления и изменения записей (INSERT, UPDATE).

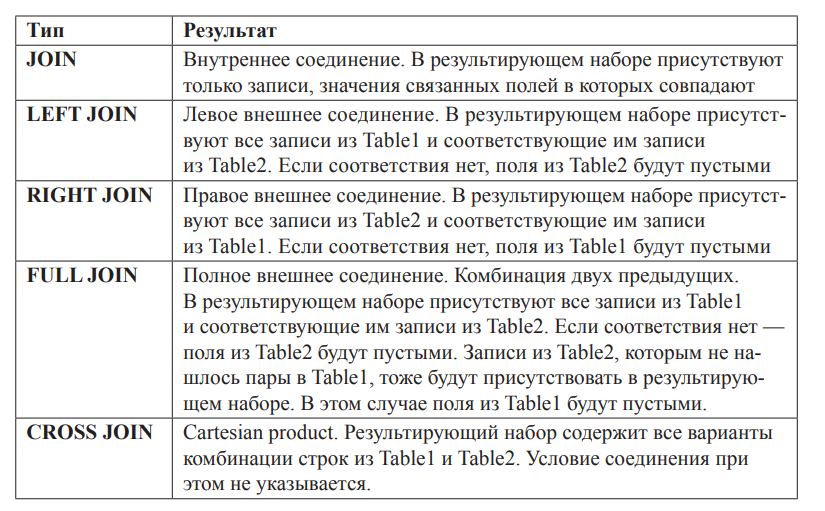




Случаи, когда стоит избегать использования индексов:

* Не стоит использовать индексы для небольших таблиц.
* Не стоит использовать индексы для таблиц, в которых, как предполагается, будут часто добавляться новые данные, либо эти данные будут изменяться.
* Не стоит использовать индекс для колонок, с которыми будут производиться частые манипуляции.
* Не стоит использовать индексы для колонок, которые имеют много значений NULL.

1. Какие есть типы JOIN'ов. Кратко опишите каждый из типов.



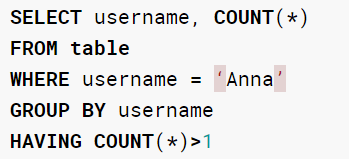
1. Для чего используется слово HAVING? Отличия от WHERE.

Во-первых, в **HAVING** и только в нём можно писать условия по агрегатным функциям (SUM, COUNT, MAX, MIN и т. д.). То есть если вы хотите сделать что-то вроде COUNT(\*) > 10, то это возможно сделать только в **HAVING**.

"Почему бы не оставить только **HAVING**?" - спросите вы. Всё кроется в том, как **SQL Server** выполняет запрос, в каком порядке происходит его разбор и работа с данными. **WHERE** выполняется до формирования групп **GROUP BY**. Это нужно для того, чтобы можно было оперировать как можно меньшим количеством данных и сэкономить ресурсы сервера и время пользователя.

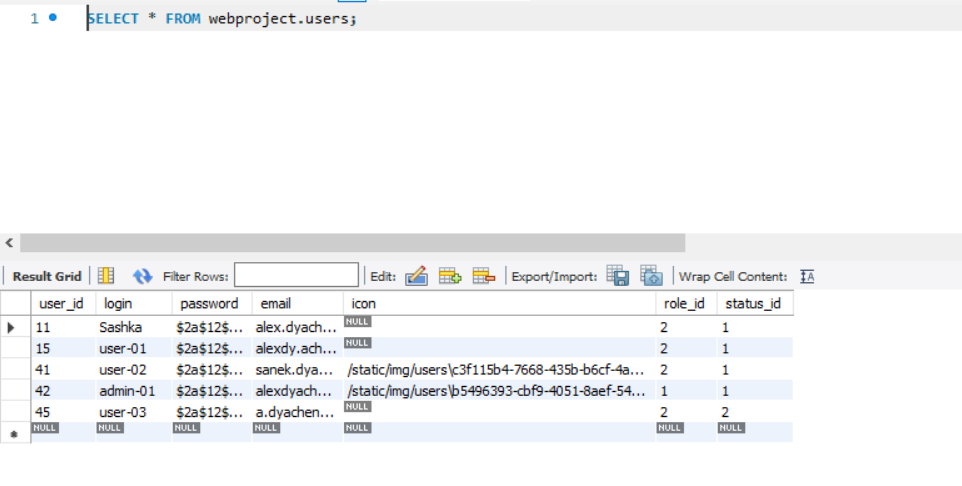
Следующим этапом формируются группы, которые указаны в **GROUP BY**. После того как сформированы группы, можно накладывать условия на результаты агрегатных функций. И тут как раз наступает очередь **HAVING**: выполняются условия, которые вы задали.

Главное отличие **HAVING** от **WHERE** в том, что в **HAVING** можно наложить условия на результаты группировки, потому что порядок исполнения запроса устроен таким образом, что на этапе, когда выполняется **WHERE**, ещё нет групп, а **HAVING** выполняется уже после формирования групп.



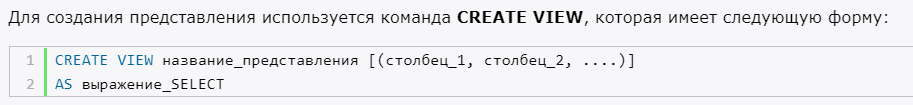
1. Зачем нужно View и какие поля там будут.

Пример:



Представления или Views представляют виртуальные таблицы. Но в отличии от обычных стандартных таблиц в базе данных представления содержат запросы, которые динамически извлекают используемые данные.

Представления дают нам ряд преимуществ. Они упрощают комплексные SQL-операции. Они защищают данные, так как представления могут дать доступ к части таблицы, а не ко всей таблице. Представления также позволяют возвращать отформатированные значения из таблиц в нужной и удобной форме.

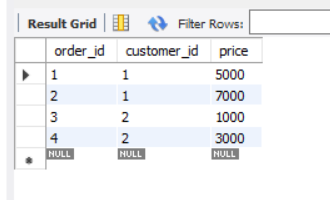


1. Есть таблицы Customer и Order. Вывести всех сustomer, у которых суммарный заказ будет > 10000.

SELECT customer.login FROM customer

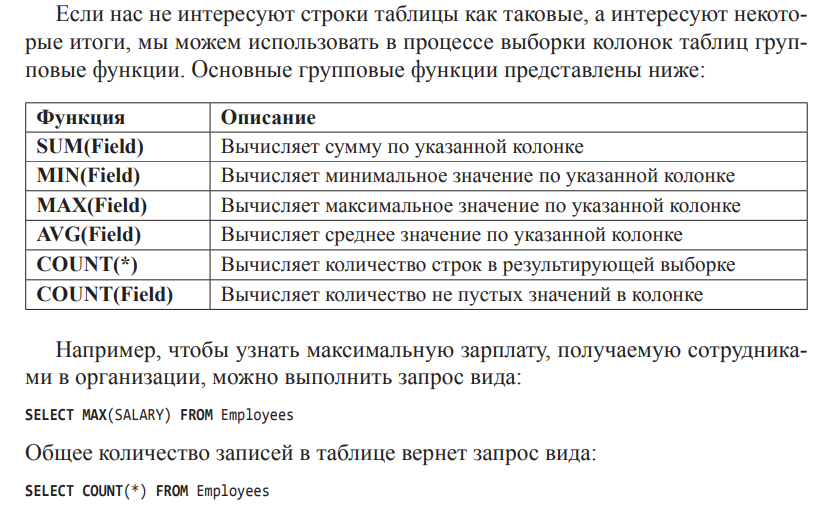
JOIN orders ON customer.customer\_id = orders.customer\_id

GROUP BY customer.login HAVING SUM(orders.price) > 10000



1. Что такое агрегирующая функция, примеры.

[Агрегирующие функции](http://dspace.ut.ee/bitstream/handle/10062/10137/_4.html)используются для получения обобщающих значений. Их можно применять ко всем строкам таблицы, строкам, определенным в предложении WHERE или к группам строк в предложении GROUP BY. В любом случае, независимо от структуры набора строк, для каждого из них получается единственное значение.



1. Результат запроса *Select \* from Table1, Table2*;
2. Есть 2 таблицы

|  |  |  |
| --- | --- | --- |
| IDDepartment | Department |  |
|  | | |
| IDEmployer | Salary | IDDepartment |

- Написать запросы на выборку Department, salary если сумма salary>100;

- Написать запрос на ту же выборку через join;