# Базы данных и SQL

## Лекция 18

Адукар

# Что изучим сегодня

- Первичным и ссылочный ключ
- СУБД
- Реляционные базы данных
- Операторы для запросов
- Логика написания запросов

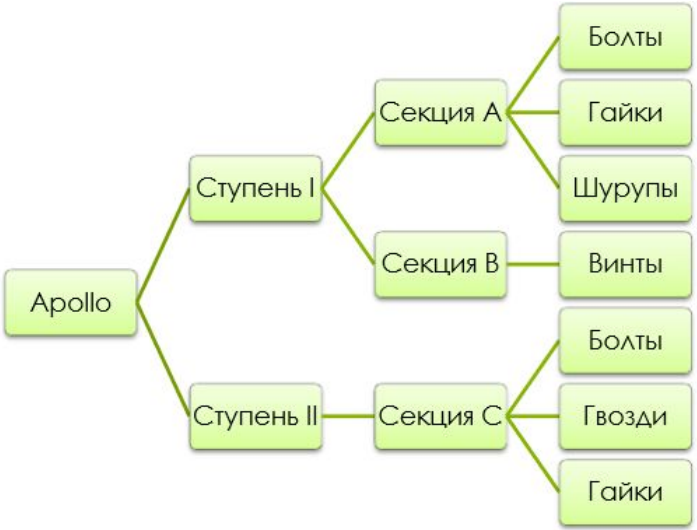
# История баз данных

История активного развития баз данных начинается с одного из самых значительных и неоднозначных событий: полета на Луну. Тогда для участия в проекте Apollo правительством США была привлечена компания Rockwell. Для того, чтобы построить космический корабль, как многие, наверное, догадываются, нужно собрать несколько миллионов деталей. И в те далекие времена была создана система управления файлами, которая отслеживала информацию о каждой детали.

Но когда решили проверить эту систему, то обнаружили, что данные в ней повторяются по нескольку раз. Налицо была огромная избыточность. К сотрудничеству была привлечена небезызвестная IBM, и в 1968 году была разработана автоматизированная система IMS. Кардинальным нововведением было разделение функций деловой логики и данных, т.е. программисты смогли работать с информацией на логическом уровне, при этом база данных выполняла функцию физического хранения. Это позволило существенно повысить производительность труда. Данная система имела следующую модель хранения данных (на рисунке представлена абстракцией!):

# Иерархическая модель

Поле Name в такой модели содержало название отдельно взятого элемента, поле ID было введено для уникальной идентификации, а колонка Id_Parent – для определения связей элементов друг с другом




| Id | Name | Id_Parent |
|---|---|---|
| 1 | Apollo | NULL |
| 2 | Ступень I | 1 |
| 3 | Ступень II | 1 |
| 4 | Секция A | 2 |
| 5 | Секция B | 2 |
| 6 | Секция C | 3 |
| 7 | Болты | 4 |
| 8 | Гайки | 4 |
| 9 | Шурупы | 4 |
| 10 | Винты | 5 |
| 11 | Болты | 6 |
| 12 | Гвозди | 6 |
| 13 | Гайки | 6 |

Таким образом, благодаря иерархической модели, было выведено несколько фундаментальных понятий:

Первичный ключ (Primary Key [PK]) – ограничение, позволяющее однозначно идентифицировать каждую запись в таблице. Первичный ключ должен содержать уникальные значения и не может быть пуст.

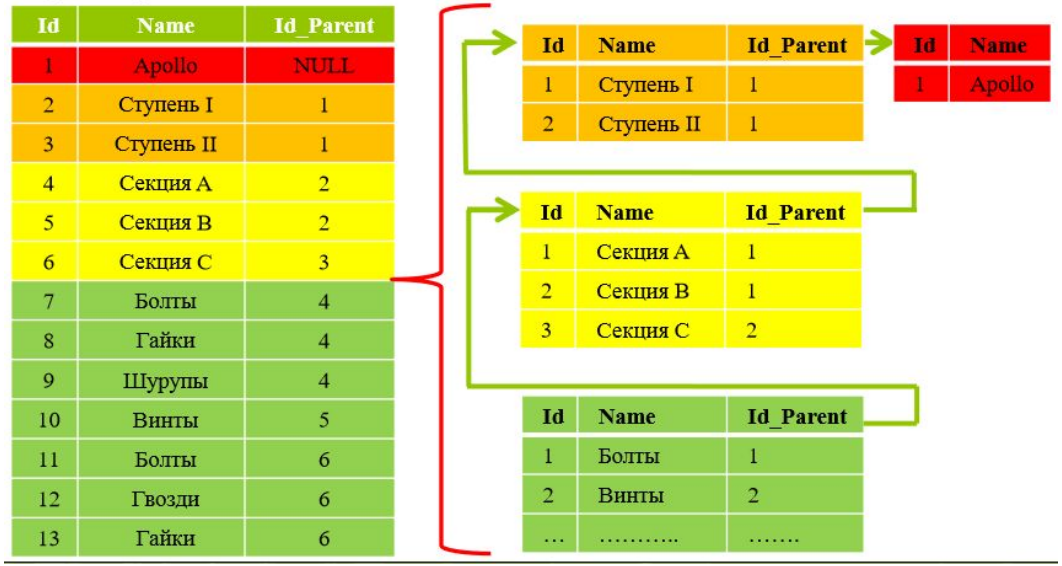
Ссылочный ключ (Foreign Key [FK]) – это ограничение, позволяющее ссылаться на какой-либо Primary Key. Зачастую, может быть неуникален и содержать пустые множества.

В примере, приведённом ранее, Id – это первичный ключ. Id_Parent – ссылочный ключ.

Следующий большой шаг в истории развития баз данных сделал доктор Эдгар Кодд (Edgar Codd) - научный сотрудник все той же самой небезызвестной IBM. В 1970 году он опубликовал свою работу "Реляционная модель для больших банков совместно используемых данных", которая в корне изменила теорию баз данных. Он предложил избавиться от иерархий и преобразовать их в разделение на логические сущности. Для каждой такой сущности подразумевалось наличие отдельной таблицы и связей между ними.

# Реляционная модель данных

В реляционной модели, придуманной Коддом, данные можно было свободно описывать в их естественном виде, без каких-либо ограничений, которые накладываются средой физического хранения. Главная особенность такой модели – зависимость всех таблиц друг от друга.



| Id | Name | Id_Parent |
|---|---|---|
| 1 | Apollo | NULL |
| 2 | Ступень I | 1 |
| 3 | Ступень II | 1 |
| 4 | Секция A | 2 |
| 5 | Секция B | 2 |
| 6 | Секция C | 3 |
| 7 | Болты | 4 |
| 8 | Гайки | 4 |
| 9 | Шурупы | 4 |
| 10 | Винты | 5 |
| 11 | Болты | 6 |
| 12 | Гвозди | 6 |
| 13 | Гайки | 6 |

| Id | Name | Id_Parent |
|---|---|---|
| 1 | Ступень I | 1 |
| 2 | Ступень II | 1 |

| Id | Name |
|---|---|
| 1 | Apollo |

| Id | Name | Id_Parent |
|---|---|---|
| 1 | Секция A | 1 |
| 2 | Секция B | 1 |
| 3 | Секция C | 2 |

| Id | Name | Id_Parent |
|---|---|---|
| 1 | Болты | 1 |
| 2 | Винты | 2 |
| … | …….. | ……. |

# Основные понятия

**База данных** — набор логически связанных данных, предназначенный для удовлетворения информационных потребностей.

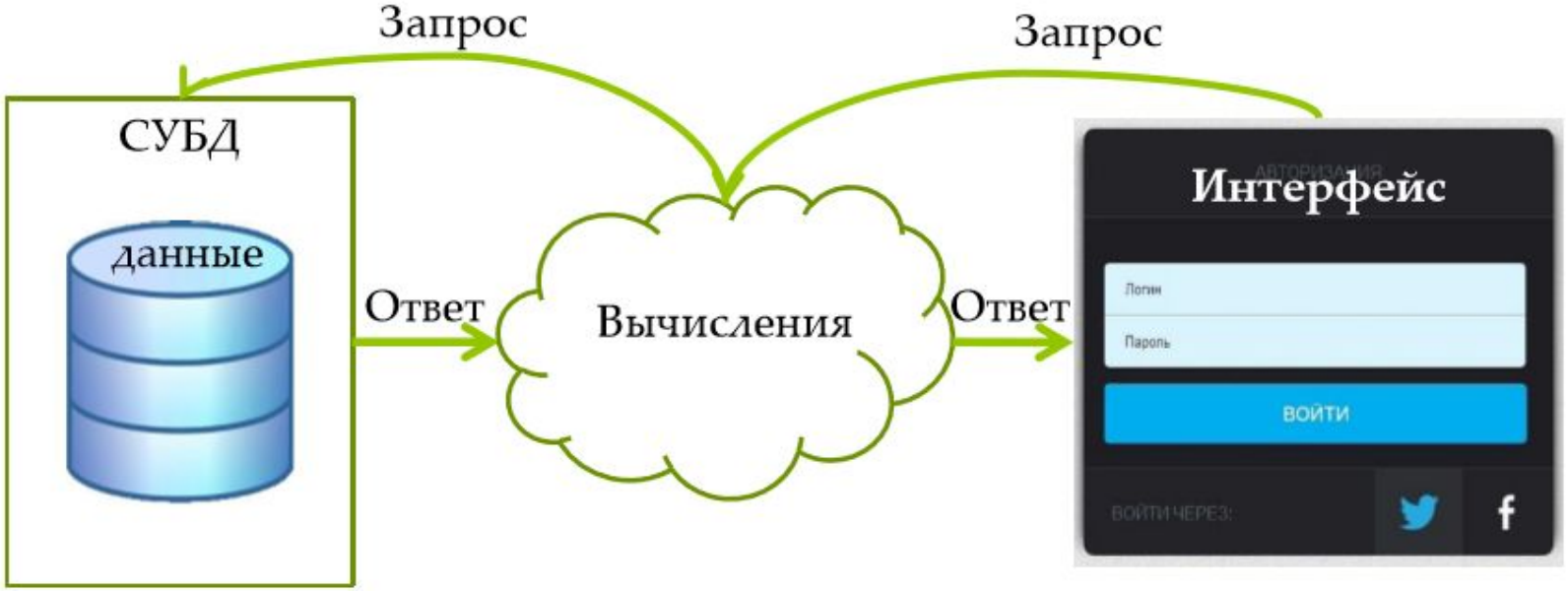
**Реляционная база данных** представляет собой множество взаимосвязанных таблиц, каждая из которых содержит информацию об объектах определенного вида. Каждая строка таблицы содержит данные об одном объекте, а столбцы таблицы содержат различные характеристики этих объектов - атрибуты.

**Система Управления Базами Данных (СУБД)** – программный комплекс для создания и администрирования (управления) базами данных.

**Data Manipulation Language** (DML) (язык управления (манипулирования) данными) — это семейство компьютерных языков, используемых в компьютерных программах или пользователями баз данных для получения, вставки, удаления или изменения данных в базах данных.

# Распространённые СУБД

Oracle (~60% рынка СУБД);
MS SQL Server (~15%);
IBM DB (~13%);
MySQL, PostgreSQL, SQLite, Sybase, Access и другие (~12%);


Запрос
Запрос
СУБД
данные
Ответ
Вычисления
Ответ
Интерфейс
Логин
Пароль
ВОЙТИ
ВОЙТИ ЧЕРЕЗ:

# Подмножества запросов

**Structured Query Language – структурированный язык запросов, предназначенный для работы с данными в рамках СУБД. Язык SQL состоит из нескольких основных подмножеств:**

**1. Data Definition Language (язык определения данных) :**

**a) CREATE (создать сущность);**

**b) ALTER (изменить сущность);**

**c) DROP (удалить сущность);**

**d) TRUNCATE (усечь сущность);**

**2. Data Manipulation Language (язык управления данными) :**

**a) SELECT (выборка данных);**

**b) UPDATE (обновление данных);**

**c) INSERT (вставка данных);**

**d) DELETE (удаление данных);**

**3. Data Control Language (язык контроля данных) :**

**a) GRANT (предоставить привилегии на какую-либо команду какому-либо пользователю);**

**b) REVOKE (изъять привилегии у пользователя).**

**4. Transaction Control Language (язык управления транзакциями) :**

**a) COMMIT применяет транзакцию;**

**b) ROLLBACK «откатывает» все изменения, сделанные в контексте текущей транзакции;**

**c) SAVEPOINT делит транзакцию на более мелкие участки.**

# SQL. Выборка данных

**Программный код на языке SQL пишется в виде запросов к данным. Сам по себе язык является регистронезависимым, но стандарт языка рекомендует писать ключевые слова SQL в запросе прописными буквами или, хотя бы, в одном стиле. Общий вид запроса на выборку данных:**

```
SELECT имя_поля_1, имя_поля_2, …
  FROM название_таблицы
WHERE имя_поля_1 = …
AND/OR имя_поля_2 = …
```

**Например, нам нужно выбрать все записи из таблицы table_Section, которая имеет вид:**

| Id | Name | Id_Parent |
|---|---|---|
| 1 | Секция A | 1 |
| 2 | Секция B | 1 |
| 3 | Секция C | 2 |

**где id и id_Parent – целочисленные поля типа int, а Name – строчное поле типа varchar(30).**

Для выборки всех данных из  таблицы, напишем запрос:

```
SELECT *
FROM table_Section
```

‘SELECT’ – это команда выборки. Затем, вместо явного указания имён колонок через запятую, пишем ‘*’ (говорится как «звёздочка») – это символ, указывающий на выборку всех полей таблицы,
‘FROM’ – команда, которая указывает из какой именно сущности будет производиться выборка, и ‘table_Section’ – название сущности. Т.е., говоря по-русски, мы сообщаем СУБД, что хотим видеть выборку всех данных из таблицы table_Section.

Попробуем немного усложнить задачу, добавив условие. Допустим, теперь нам нужно вычленить записи с id больше, либо равным двум. Запрос будет иметь вид:

```
SELECT *
FROM table_Sections
WHERE id >= 2
```

| Id | Name | Id_Parent |
|---|---|---|
| 1 | Секция A | 1 |
| 2 | Секция B | 1 |
| 3 | Секция C | 2 |

Результат:

| Id | Name | Id_Parent |
|---|---|---|
| 2 | Секция B | 1 |
| 3 | Секция C | 2 |

Мы сказали этим запросом, что хотим вычленить все данные, у которых ('WHERE' – ключевое слово, начало блока условий) поле id принимает значения от двух и выше.

**Продолжаем изучение языка SQL и его логических операторов. Допустим, у нас есть таблица с названием Employee вида:**

| id | name | org | salary |
|---|---|---|---|
| 1 | Гришин | Бухгалтерия | 10000 |
| 2 | Васильев | IT | 15000 |
| 3 | Петров | Снабжение | 7500 |
| 4 | Иванов | IT | 20000 |

**Основным оператором фильтрации данных служит ключевое слово AND, позволяющее накладывать несколько условий на выборку. Таким образом, данные должны удовлетворять всем (!) условиям, указанным в WHERE. Например, нам нужно вычленить всех IT'шников с зарплатой, больше 16 тысяч. Для этого пишем запрос вида:**

```
SELECT name
FROM Employee
WHERE org = 'IT' AND salary > 16000;
```

**Запрос выдает нам следующее:**

| name |
|---|
| Иванов |

Обратите внимание на этот запрос, который мы только что написали:

```
SELECT name
FROM Employee
 WHERE org = ‘IT’
AND salary > 16000;
```

В нём присутствует несколько важных аспектов. Во-первых, условие всегда начинается с блока WHERE, но данный оператор пишется в одном запросе лишь единожды. После него всегда пишется имя поля, на которое необходимо добавить условие, затем символ логики (‘>’ – строго больше, ‘<’ – строго меньше, ‘>=’ – больше или равно, ‘<=’ – меньше или равно, ‘=’ – строго равно, ‘!=’ или ‘<>’ – не равно) и только потом значение.
Во-вторых, числовые данные пишутся без кавычек, а строчные всегда в одинарных кавычках. **Это очень важный момент!**
В третьих, для начала нового условия применяется вспомогательный оператор AND. Никаких WHERE повторно не пишется, только дополнительный предикат!

**Таблица остаётся той же. Теперь нас интересуют все бухгалтеры или снабженцы с зарплатой менее 8000. В этом нам поможет оператор OR, благодаря которому можно указать несколько условий и выборка должна удовлетворять хотя бы одному из них:**

```
SELECT name
FROM Employee
WHERE org = 'Бухгалтерия'
OR (org = 'Снабжение'
AND salary < 8000);
```

**Запрос выдаст нам следующее:**

| name |
|---|
| Гришин |
| Петров |

| id | name | org | salary |
|---|---|---|---|
| 1 | Гришин | Бухгалтерия | 10000 |
| 2 | Васильев | IT | 15000 |
| 3 | Петров | Снабжение | 7500 |
| 4 | Иванов | IT | 20000 |

**Не сложно заметить, что первая строка выборки удовлетворяет одному условию, а другая – другому. Обратите внимание на группировку условий в скобках**

# SQL. Логика. BETWEEN и DISTINCT

**Когда мы работаем с числовым или табельным диапазоном значений, часто приходится прибегать к написанию интервальных запросов. В этом нам поможет оператор BETWEEN. Давайте выделим из таблицы все организации, которые платят от 10 до 30 тысяч:**

| id | name | org | salary |
|---|---|---|---|
| 1 | Гришин | Бухгалтерия | 10000 |
| 2 | Васильев | IT | 15000 |
| 3 | Петров | Снабжение | 7500 |
| 4 | Иванов | IT | 20000 |

```
SELECT DISTINCT org    ---- DISTINCT исключает повторы!
FROM Employee
WHERE salary BETWEEN 10000 AND 30000;
```

**Запрос выдаст нам следующее:**

| org |
|---|
| Бухгалтерия |
| IT |

**Важно: границы диапазона в BETWEEN будут включены в выборку! Обратите внимание на вспомогательный оператор DISTINCT. Для того, чтобы исключить дубликаты (например, чтобы запись IT не повторялась), мы написали его перед именем поля и после оператора SELECT (строго так!). В итоге, получили уникальную выборку.**

# SQL. LIKE

**Очень часто возникает ситуация, когда мы хотим сделать выборку, но точных условий задать не можем. Например, когда хотим выбрать данные по сотруднику, но помним лишь первые несколько букв его фамилии. Для решения таких задач нам поможет оператор LIKE. Допустим, мы помним, что фамилия нужного нам человека начиналась с «Ва»:**

| id | name | org | salary |
|---|---|---|---|
| 1 | Гришин | Бухгалтерия | 10000 |
| 2 | Васильев | IT | 15000 |
| 3 | Петров | Снабжение | 7500 |
| 4 | Иванов | IT | 20000 |

```
SELECT *
FROM Employee
WHERE name LIKE 'Ва%';
```

**Результат запроса:**

| id | name | org | salary |
|---|---|---|---|
| 2 | Васильев | IT | 15000 |

**Символ ‘%’ означает, что после букв ‘Ва’ могут быть какие угодно символы. Их количество от 0 до N. Важно понимать, что этими символами могут быть цифры, буквы и т.д. Знак % можно использовать несколько раз. Например, так выглядит запрос, который выводит данные о сотрудниках, у которых в зарплате содержится цифра 500:**

| id | name | org | salary |
|---|---|---|---|
| 1 | Гришин | Бухгалтерия | 10000 |
| 2 | Васильев | IT | 15000 |
| 3 | Петров | Снабжение | 7500 |
| 4 | Иванов | IT | 20000 |

```
SELECT*
FROM Employee
WHERE salary LIKE ‘%500%’;
```

**Результат запроса:**

| id | name | org | salary |
|---|---|---|---|
| 2 | Васильев | IT | 15000 |
| 3 | Петров | Снабжение | 7500 |

# SQL. Логика. IN/NOT IN

**Когда условий становится много и они накладываются на одно поле, можно совместить их множеством, а выборку с оператора OR изменить на конструкцию IN или NOT IN.**

**К примеру, нас интересуют все сотрудники из отделов IT и Бухгалтерии:**

```
SELECT name
FROM Employee
WHERE org IN ('Бухгалтерия', 'IT');
```

| id | name | org | salary |
|---|---|---|---|
| 1 | Гришин | Бухгалтерия | 10000 |
| 2 | Васильев | IT | 15000 |
| 3 | Петров | Снабжение | 7500 |
| 4 | Иванов | IT | 20000 |

**Запрос выдаст нам следующее:**

| name |
|---|
| Гришин |
| Васильев |
| Иванов |

**Как не сложно догадаться, оператор IN работает похожим на OR способом, т.е., в блоке WHERE можно было бы написать “org = ‘Бухгалтерия’ OR org = ‘IT’”, но если условий много, например, 10 или 20?**
**В таких случаях и помогает IN, который проверяет поле на каждое из значений. Совпадение найдено? Значит результат попадёт в выборку.**

**В предыдущем запросе заменим IN на NOT IN. Получим:**

```
SELECT name
FROM Employee
WHERE org NOT IN (‘Бухгалтерия’, ‘IT’);
```

| id | name | org | salary |
|---|---|---|---|
| 1 | Гришин | Бухгалтерия | 10000 |
| 2 | Васильев | IT | 15000 |
| 3 | Петров | Снабжение | 7500 |
| 4 | Иванов | IT | 20000 |

**Результат:**

| name |
|---|
| Петров |

**Запрос вернул нам все значения name, которые НЕ(!) попали во множество (‘Бухгалтерия’, ‘IT’).**

# SQL. Логика. Подзапросы

**Очень часто мы сталкиваемся с ситуацией, когда в явном виде не знаем условий, но знаем как эти самые условия получить запросом. Такие ситуации порождают использование подзапросов. Например, нужно извлечь сотрудников, которые работают в IT, получают зарплату меньше 20 тысяч и идентификатор у них больше идентификатора Гришина. Вот это самое важное. Как такое написать?**

**Пробуем:**

```
SELECT name
 FROM Employee WHERE org = 'IT'
AND salary < 20000  AND id >
(SELECT id FROM Employee WHERE name = 'Гришин');
```

| name |
| --- |
| Васильев |

**Обратите внимание: подзапрос содержится в круглых скобках. С его помощью мы находим идентификатор работника Гришина (подзапрос найдёт единицу) и это значение будет подставлено в основной запрос в виде «id > 1». Таким образом, не зная явного значения id_Гришина, мы написали дополнительный запрос на извлечение ячейки как условия**

# SQL. NULL

**NULL в СУБД — специальное значение (псевдозначение), которое может быть записано в поле таблицы базы данных (БД). NULL соответствует понятию «пустое поле», то есть «поле, не содержащее никакого значения» / пустое множество. Введено для того, чтобы различать в полях БД пустые (визуально не отображаемые) значения (например, строку нулевой длины) и отсутствующие значения (когда в поле не записано вообще никакого значения, даже пустого). NULL означает отсутствие, неизвестность информации. Значение NULL не является значением в полном смысле слова: по определению оно означает отсутствие значения и не принадлежит ни одному типу данных. Поэтому NULL не равно ни логическому значению FALSE, ни пустой строке, ни нулю. При сравнении NULL с любым значением будет получен результат NULL, а не FALSE и не 0. Более того, NULL не равно NULL! Представим себе ситуацию, когда Петрову решили изменить заработную плату, но пока не определились с конечной цифрой. В итоге, у него может быть в поле Salary всё что угодно – хоть 0, хоть миллион баксов, но пока неизвестно. А теперь попытаемся найти всех работников, у которых показатель заработной платы был бы пуст / неизвестен:**

```
SELECT name
FROM Employees
WHERE salary IS NULL;
```

| name |
|---|
| Петров |

| id | name | org | salary |
|---|---|---|---|
| 1 | Гришин | Бухгалтерия | 10000 |
| 2 | Васильев | IT | 15000 |
| 3 | Петров | Снабжение | **NULL** |
| 4 | Иванов | IT | 20000 |

**Обратите внимание: в проверке никаких знаков равенства! Только IS NULL!**

# Что изучили сегодня

- Первичный и ссылочный ключ
- СУБД
- Реляционные базы данных
- Логика написания запросов
- Where/ Like/ In or Not in/ Between/ Distinct

# Домашнее задание

1) Грабер М. Введение в SQL – главы 1, 2, 3, 4 и 5 полностью;
2) на сайте SQL-EX.RU выполнить к следующему занятию задачи 1, 2, 3, 4, 5, 31, 33, 42.
3) в качестве справочников рекомендуется использовать http://sql-tutorial.ru/ и http://www.w3schools.com/sql/