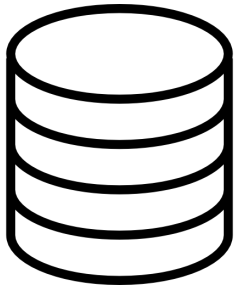
Базы данных

SQL

Базы данных

База данных (БД) - упорядоченный набор логически взаимосвязанных данных, используемых совместно, и которые хранятся в одном месте. Если коротко, то простейшая **БД** это обычная таблица со строками и столбцами в которой хранится разного рода информация (примером может служить таблица в **Excel**)



Система управления базой данных (**СУБД**) - это программные средства, которые позволяют выполнять все необходимые операции с базой данных.

СУБД - это специальная программа, которая занимается только хранением и обработкой данных, независимо от их содержания, и могут применяться в самых разных задачах.

Базы данных

Как правило, в современных информационных системах используют удалённые базы данных, расположенные на серверах (специально выделенных компьютерах) локальной или глобальной сети. В этом случае несколько пользователей могут одновременно работать с базой и вносить в неё изменения.

СУБД, работающие с удалёнными базами данных, можно разделить на два типа по способу работы с файлами:

- файл-серверные СУБД;
- клиент-серверные СУБД

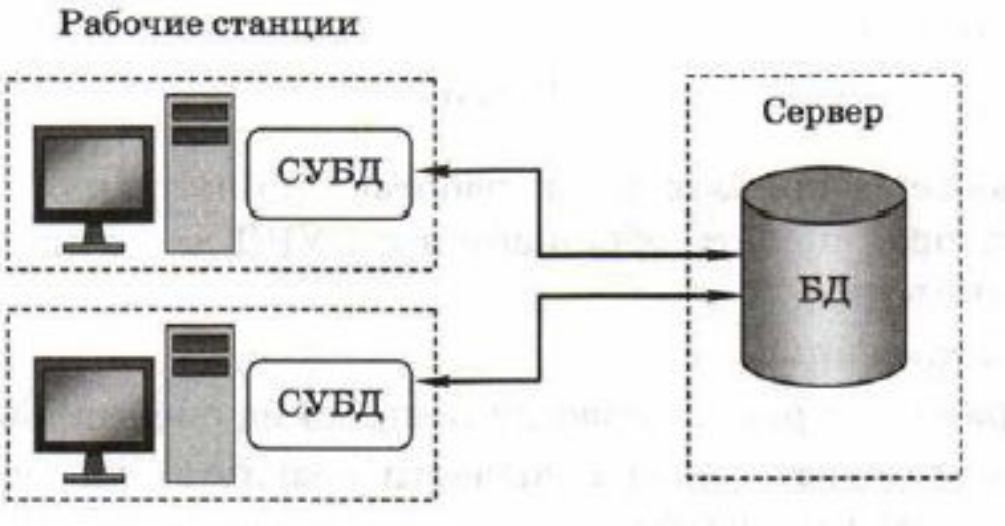
Файл-серверные СУБД

Файл-серверные СУБД (например, *Microsoft Access*) работают на компьютерах пользователей (они называются рабочими станциями). Это значит, что сервер только хранит файлы, но не участвует в обработке данных. Когда пользователь вносит изменения в базу, СУБД с его рабочей станции блокирует файлы на сервере, чтобы их не могли изменить другие пользователи.

При большом количестве пользователей проявляются недостатки файл-серверных ИС:

- обработку данных выполняют СУБД на рабочих станциях, поэтому они должны быть достаточно мощными;
- при поиске данных вся БД копируется по сети на компьютер пользователя, это создает значительную лишнюю нагрузку на сеть;
- слабая защита от неправомерного доступа к данным;
- ненадёжность при большом количестве пользователей, особенно если они вносят изменения в базу данных.

Файл-серверные СУБД



Клиент-серверные СУБД

Чтобы избавиться от этих недостатков, нужно переместить СУБД на сервер.

Клиент-серверная СУБД расположена на том же компьютере, где находится база данных. Она полностью берёт на себя всю работу с данными.

На компьютере пользователя работает прикладная программа-клиент, которая по сети обращается к СУБД для выполнения операций с данными.

Задачи клиента:

- отправить серверу команду (запрос) на специальном языке;
- получив ответ сервера, вывести результат на экран пользователя или на печать.

Клиент-серверные СУБД

Клиент-серверные СУБД

Самые известные среди коммерческих клиент-серверных СУБД - *Microsoft SQL Server u Oracle*.

Существуют и бесплатные клиент-серверные СУБД: *Firebird, PostgreSQL, MySQL*.

Достоинства клиент-серверных СУБД:

- основная обработка данных выполняется на сервере, поэтому рабочие станции могут быть маломощными;
- проще модернизация (достаточно увеличить мощность сервера);
- надёжная защита данных (устанавливается на сервере);
- снижается нагрузка на сеть, так как передаются только нужные данные (запросы и результаты выполнения запросов);
- надёжная работа при большом количестве пользователей (запросы ставятся в очередь).

Их недостатки - повышенные требования к мощности сервера и высокая стоимость коммерческих СУБД (Microsoft SQL Server и Oracle).

Клиент-серверные СУБД

В современных клиент-серверных СУБД для управления данными чаще всего используют язык **SQL** (Structured Query Language - язык структурных запросов). Он содержит все команды, необходимые работы с данными, включая получение нужной информации, создание и изменение базы данных.

Задачи сервера:

• ожидать запросы клиентов по сети;

• при поступлении запроса поставить его в очередь на выполнение;

Реляционные базы данных

Реляционная база данных – это **набор данных с предопределенными связями между ними.**

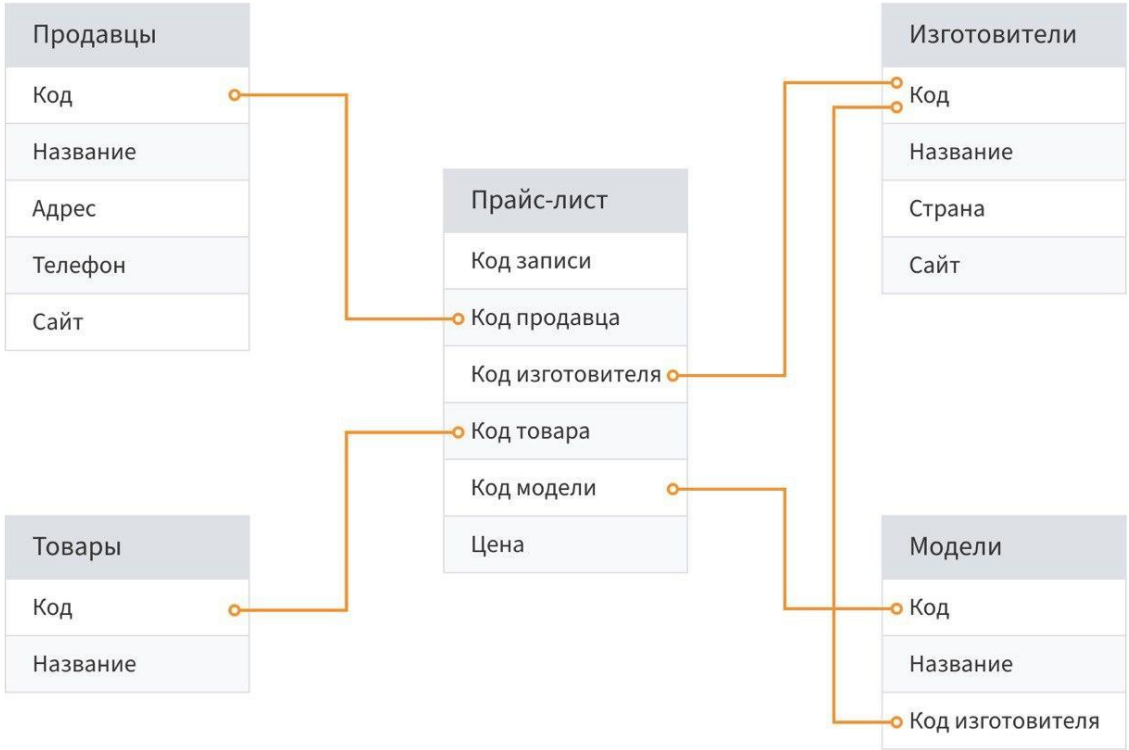
Эти данные организованы в виде набора таблиц, состоящих из столбцов и строк.

В таблицах хранится информация об объектах, представленных в базе данных.

В каждом столбце таблицы хранится определенный тип данных, в каждой ячейке – значение атрибута. Каждая строка таблицы представляет собой набор связанных значений, относящихся к одному объекту или сущности.

Каждая строка в таблице может быть помечена уникальным идентификатором, называемым первичным ключом, а строки из нескольких таблиц могут быть связаны с помощью внешних ключей. К этим данным можно получить доступ многими способами, и при этом реорганизовывать таблицы БД не требуется.

Реляционные базы данных



Реляционные базы данных

Таблицы в реляционных базах данных обладают рядом свойств. Основными являются следующие:

- В таблице не может быть двух одинаковых строк. В математике таблицы, обладающие таким свойством, называют ***отношениями*** - по-английски relation, отсюда и название - реляционные.
- Столбцы располагаются в определенном порядке, который создается при создании таблицы. В таблице может не быть ни одной строки, но обязательно должен быть хотя бы один столбец.
- У каждого столбца есть уникальное имя (в пределах таблицы), и все значения в одном столбце имеют один тип (число, текст, дата...).
- На пересечении каждого столбца и строки может находиться только атомарное значение (одно значение, не состоящее из группы значений). Таблицы, удовлетворяющие этому условию, называют ***нормализованными***.

Реляционные базы данных

Предположим, мы захотели создать базу данных для форума. У форума есть зарегистрированные пользователи, которые создают темы и оставляют сообщения в этих темах. Эта информация и должна храниться в базе данных.

Теоретически мы можем все это расположить в одной таблице, например, так:

Имя	E-mail	Пароль	Созданные темы	Созданные сообщения

Но это противоречит свойству атомарности (одно значение в одной ячейке), а в столбцах Темы и Сообщения у нас предполагается неограниченное количество значений. Значит, нашу таблицу надо разбить на три: Пользователи, Темы и Сообщения.

Пользователи

Имя	E-mail	Пароль

Темы

Наименование	Автор

Сообщения

Текст	Автор

Реляционные базы данных

Наша таблица Пользователи удовлетворяет всем условиям. А вот таблицы Темы и Сообщения - нет. Ведь в таблице не может быть двух одинаковых строк, а где гарантия, что один пользователь не оставит два одинаковых сообщения, например:

Сообщения

Текст	Автор
Думаю надо сделать так…	Кирилл
Согласен	Вася
А еще можно сделать так…	Семен
Согласен	Вася

Кроме того, мы знаем, что каждое сообщение обязательно относится к какой-либо теме. А как это можно узнать из наших таблиц? Никак. Для решения этих проблем, в реляционных базах данных существуют ***ключи***.

Реляционные базы данных

Первичный ключ **(сокращенно PK - primary key)** - столбец, значения которого во всех строках различны. Первичные ключи могут быть логическими (естественными) и суррогатными (искусственными). Так, для нашей таблицы Пользователи первичным ключом может стать столбец e-mail (ведь теоретически не может быть двух пользователей с одинаковым e-mail). На практике лучше использовать суррогатные ключи, т.к. их применение позволяет абстрагировать ключи от реальных данных. Кроме того, первичные ключи менять нельзя, а что если у пользователя сменится e-mail?

Первичный ключ представляет собой дополнительное поле в базе данных. Как правило, это порядковый номер записи (хотя вы можете задавать их на свое усмотрение, контролируя, чтобы они были уникальны).

Реляционные базы данных

Давайте внесем поля первичных ключей в наши таблицы:

Пользователи

id пользователя	Имя	E-mail	Пароль
1	Кирилл	kirill@mail.ru	Gh345fgh
2	Вася	vasy@rambler.ru	As3bh7
3	Семен	semen@yandex.ru	gk4bb6

Темы

id темы	Наименование	Автор
1	О рыбалке	Кирилл
2	Велосипеды	Вася
3	Ночные клубы	Семен
4	О рыбалке	Вася

Сообщения

id сообщения	Текст	Автор
1	Думаю надо сделать так…	Кирилл
2	Согласен	Вася
3	А еще можно сделать так…	Семен
4	Согласен	Вася

Реляционные базы данных

Теперь каждая запись в наших таблицах уникальна. Нам осталось установить соответствие между темами и сообщениями в них. Делается это так же при помощи первичных ключей. В таблицу сообщения мы добавим еще одно поле:

Сообщения

id сообщения	Текст	Автор	id темы
1	Думаю надо сделать так…	Кирилл	1
2	Согласен	Вася	4
3	А еще можно сделать так…	Семен	1
4	Согласен	Вася	1

Теперь понятно, что сообщение с id=2 принадлежит теме "О рыбалке" (id темы = 4), созданной Васей, а остальные сообщения принадлежать теме "О рыбалке" (id темы = 1), созданной Кириллом. Такое поле называется ***внешний ключ*****(сокращенно FK - foreign key).** Каждое значение этого поля соответствует какому-либо первичному ключу из таблицы "Темы". Так устанавливается однозначное соответствие между сообщениями и темами, к которым они относятся.

Реляционные базы данных

Последний нюанс. Предположим, у нас добавился новый пользователь, и зовут его тоже Вася:

Пользователи

id пользователя	Имя	E-mail	Пароль
1	Кирилл	kirill@mail.ru	*******
2	Вася	vasy@rambler.ru	*******
3	Семен	semen@yandex.ru	*******
4	Вася	vasy@mail.ru	*******

Как мы узнаем, какой именно Вася оставил сообщения? Для этого поля автор в таблицах "Темы" и "Сообщения" мы сделаем также внешними ключами:

Сообщения

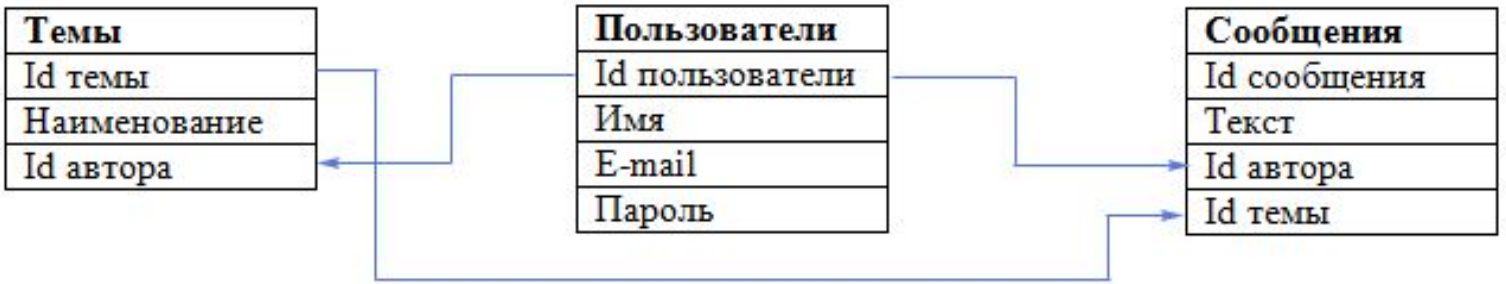
id сообщения	Текст	id автора	id темы
1	Думаю надо сделать так…	1	1
2	Согласен	2	4
3	А еще можно сделать так…	3	1
4	Согласен	2	1

Темы

id темы	Наименование	id автора
1	О рыбалке	1
2	Велосипеды	2
3	Ночные клубы	3
4	О рыбалке	1
5	К кому обратиться	4

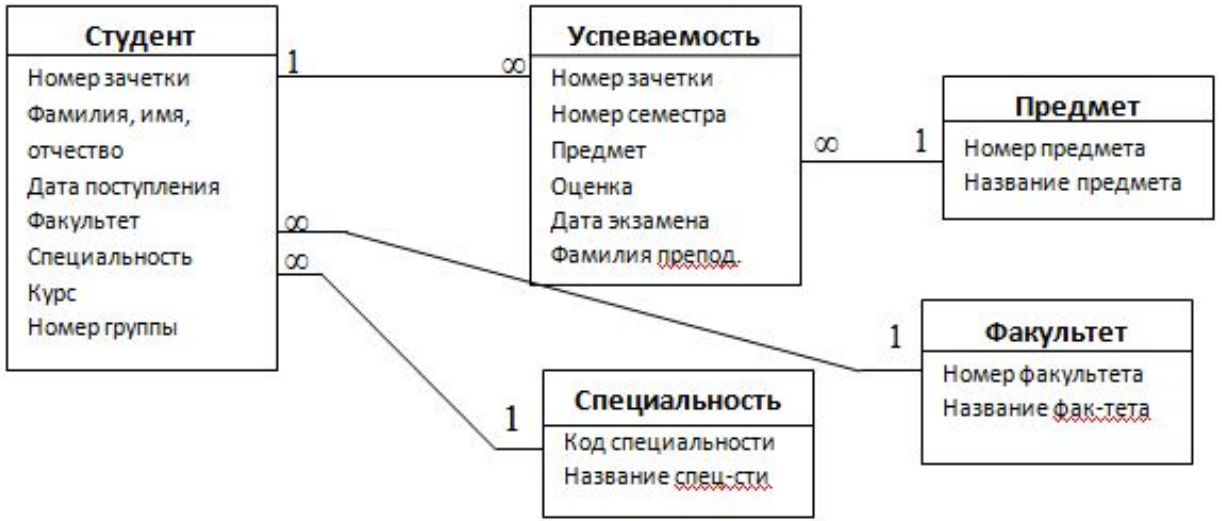
Реляционные базы данных

Наша база данных готова. Схематично ее можно представить так:



ER диаграмма

ENTITY RELATIONAL (ER) MODEL — это концептуальная модель модели данных высокого уровня. ER моделирование помогает систематически анализировать требования к данным для создания хорошо спроектированной базы данных.



SQL

SQL — декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных.

Атрибуты столбцов

PRIMARY KEY
- Атрибут PRIMARY KEY задает первичный ключ таблицы.

AUTO_INCREMENT
- Атрибут AUTO_INCREMENT позволяет указать, что значение столбца будет автоматически увеличиваться при добавлении новой строки. Данный атрибут работает для столбцов, которые представляют целочисленный тип или числа с плавающей точкой.

UNIQUE
- Атрибут UNIQUE указывает, что столбец может хранить только уникальные значения.

NULL и NOT NULL
- Чтобы указать, может ли столбец принимать значение NULL, при определении столбца ему можно задать атрибут NULL или NOT NULL. Если этот атрибут явным образом не будет использован, то по умолчанию столбец будет допускать значение NULL. Исключением является тот случай, когда столбец выступает в роли первичного ключа - в этом случае по умолчанию столбец имеет значение NOT NULL.

DEFAULT
- Атрибут DEFAULT определяет значение по умолчанию для столбца. Если при добавлении данных для столбца не будет предусмотрено значение, то для него будет использоваться значение по умолчанию.

SQL типы данных

Символьные типы

- *CHAR*: представляет стоку фиксированной длины. Длина хранимой строки указывается в скобках, например, CHAR(10) - строка из десяти символов. Строка дополняется пробелами.
- *VARCHAR*: представляет стоку переменной длины. Будет занимать именно столько места, сколько необходимо.
- *TEXT*: представляет текст длиной до 65 КБ.

Числовые типы

- *TINYINT*: представляет целые числа от -127 до 128, занимает 1 байт.
- *BOOL*: фактически не представляет отдельный тип, а является лишь псевдонимом для типа *TINYINT(1)* и может хранить два значения 0 и 1. Однако данный тип может также в качестве значения принимать встроенные константы *TRUE* (представляет число 1) и *FALSE* (предоставляет число 0).Также имеет псевдоним *BOOLEAN*.
- *INT*: представляет целые числа от -2147483648 до 2147483647, занимает 4 байта. Псевдоним – *INTEGER*.
- *FLOAT*: хранит дробные числа с плавающей точкой одинарной точности, занимает 4 байта
- *DOUBLE*: хранит дробные числа с плавающей точкой двойной точности, занимает 8 байт. Данный тип также имеет псевдонимы REAL и DOUBLE PRECISION, которые можно использовать вместо DOUBLE.

Типы для работы с датой и временем

- *DATE*: хранит даты с 1 января 1000 года до 31 деабря 9999 года (с "1000-01-01" до "9999-12-31"). По умолчанию для хранения используется формат yyyy-mm-dd. Занимает 3 байта.
- *TIME*: хранит время от -838:59:59 до 838:59:59. По умолчанию для хранения времени применяется формат "hh:mm:ss". Занимает 3 байта.
- *DATETIME*: объединяет время и дату, диапазон дат и времени - с 1 января 1000 года по 31 декабря 9999 года (с "1000-01-01 00:00:00" до "9999-12-31 23:59:59"). Для хранения по умолчанию используется формат "yyyy-mm-dd hh:mm:ss". Занимает 8 байт

SQL операторы

Операции сравнения

- = сравнение на равенство
- != сравнение на равенство
- <> сравнение на неравенство
- < меньше чем
- > больше чем
- <= меньше чем или равно
- >= больше чем или равно

Логические операторы

- AND операция логического И.
- OR операция логического ИЛИ.
- NOT операция логического отрицания.

Агрегатные функции

- AVG вычисляет среднее значение
- SUM вычисляет сумму значений
- MIN вычисляет наименьшее значение
- MAX вычисляет наибольшее значение
- COUNT вычисляет количество строк в запросе

SQL

Создание таблицы:

```
CREATE TABLE <table_name1> (
 <col_name1><col_type1>,
 <col_name2><col_type2>,
 <col_name3><col_type3>
 PRIMARY KEY(<col_name1>),
 FOREIGN KEY(<col_name2>) REFERENCES <table_name2>(<col_name2>)
 );
```

Удаление таблицы:

```
DROP TABLE <table_name>
```

SQL

Вставка записи в таблицу:

Команда INSERT INTO <table_name> в SQL отвечает за добавление данных в таблицу:

```
INSERT INTO <table_name> (<col_name1>, <col_name2>, <col_name3>, …)
 VALUES (<value1>, <value2>, <value3>, …);
```

При добавлении данных в каждый столбец таблицы не требуется указывать названия столбцов.

```
INSERT INTO <table_name>
 VALUES (<value1>, <value2>, <value3>, …);
```

Обновление данных:

```
UPDATE <table name>
 SET <col_name1> = <value1>, <col_name2> = <value2>, ...
  WHERE <condition>;
```

SQL

Выбор записей:

```
SELECT <col_name1>, <col_name2>, …
 FROM <table_name>;
```

Выбрать все записи из таблицы:

```
SELECT * FROM <table_name>;
```

Получение только неповторяющихся записей:

```
SELECT DISTINCT <col_name1>, <col_name2>, …
 FROM <table_name>;
```

Выбор записей с условием:

```
SELECT <col_name1>, <col_name2>, …
 FROM <table_name>
 WHERE <condition>;
```

SQL

Группировка:

Оператор GROUP BY часто используется с агрегатными функциями, такими как COUNT, MAX, MIN, SUM и AVG, для группировки выходных значений.

```
SELECT <col name1>, <col_name2>, …
 FROM <table_name>
  GROUP BY <col_namex>;
```

Группировка записей по условию:

Ключевое слово HAVING было добавлено в SQL по той причине, что WHERE не может использоваться для работы с агрегатными функциями.

```
SELECT <col_name1>, <col_name2>, ...
 FROM <table_name>
 GROUP BY <column_namex>
 HAVING <condition>
```

SQL

Сортортировка:

ORDER BY используется для сортировки результатов запроса по убыванию или возрастанию. ORDER BY отсортирует по возрастанию, если не будет указан способ сортировки ASC или DESC.

```
SELECT <col_name1>, <col_name2>, …
 FROM <table_name>
 ORDER BY <col_name1>, <col_name2>, … ASC|DESC;
```

Выбор из промежутка значений:

BETWEEN используется для выбора значений данных из определённого промежутка. Могут быть использованы числовые и текстовые значения, а также даты.

```
SELECT <col_name1>, <col_name2>, …
 FROM <table_name>
 WHERE <col_namex> BETWEEN <value1> AND <value2>;
```

SQL

Шаблон поиска:

Оператор LIKE используется в WHERE, чтобы задать шаблон поиска похожего значения.

Есть два свободных оператора, которые используются в LIKE:

- % (ни одного, один или несколько символов);
- _ (один символ).

```
SELECT <col_name1>, <col_name2>, …
 FROM <table_name>
 WHERE <col_namex> LIKE <pattern>;
```

Вхождение в:

С помощью IN можно указать несколько значений для оператора WHERE:

```
SELECT <col_name1>, <col_name2>, …
 FROM <table_name>
 WHERE <col_namen> IN (<value1>, <value2>, …);
```

SQL

Выбор из промежутка значений:

BETWEEN используется для выбора значений данных из определённого промежутка. Могут быть использованы числовые и текстовые значения, а также даты.

```
SELECT <col_name1>, <col_name2>, …
 FROM <table_name>
 WHERE <col_namex> BETWEEN <value1> AND <value2>;
```

Оператор JOIN

Оператор JOIN используется для **соединения двух или нескольких таблиц**. Соединение таблиц может быть внутренним (**INNER**) или внешним (**OUTER**), причем внешнее соединение может быть левым (**LEFT**), правым (**RIGHT**) или полным (**FULL**). Далее на примере двух таблиц рассмотрим различные варианты их соединения.

Синтаксис соединения таблиц оператором JOIN имеет вид:

```
FROM <таблица 1>
 [INNER]
 {{LEFT | RIGHT | FULL } [OUTER]} JOIN <таблица 2>
[ON <предикат>]
```

Предикат в этой конструкции определяет условие соединения строк из разных таблиц.

Оператор JOIN

Допустим есть две таблицы (Auto слева и Selling справа), в каждой по четыре записи. Одна таблица содержит названия марок автомобилей (Auto), вторая количество проданных автомобилей (Selling):

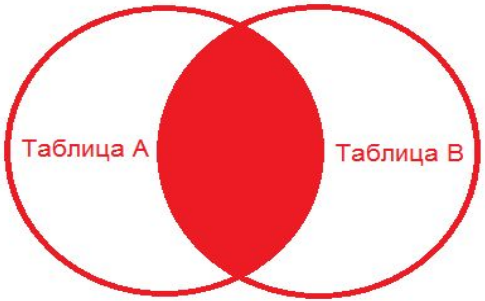
id	name
1	bmw
2	opel
3	kia
4	audi

id	sum
1	250
5	450
3	300
6	400

Далее соединим эти таблицы по полю id несколькими различными способами. Совпадающие значения выделены цветом для лучшего восприятия.

Оператор INNER JOIN

Внутреннее соединение (**INNER JOIN**) означает, что в результирующий набор попадут только те соединения строк двух таблиц, для которых значение предиката равно TRUE. Обычно используется для **объединения записей, которые есть и в первой и во второй таблице**, т. е. получения **пересечения таблиц**:



Красным выделена область, которую мы должны получить.

Итак, сам запрос:

```
SELECT * FROM 'Auto'
 INNER JOIN 'Selling' ON 'Auto'.id = 'Selling'.id
```

id	name	id	sum
1	bmw	1	250
3	kia	3	300

*Ключевое слово INNER в запросе можно опустить.

В итоге запрос отбирает и соединяет те записи, у которых значение поля id в обоих таблицах совпадает.

Оператор FULL JOIN

Внешнее соединение (**OUTER JOIN**) бывает нескольких видов. Первым рассмотрим **полное внешнее объединение** (**FULL OUTER JOIN**), которое объединяет записи из обеих таблиц (если условие объединения равно true) и дополняет их всеми записями из обеих таблиц, которые не имеют совпадений. Для записей, которые не имеют совпадений из другой таблицы, недостающее поле будет иметь значение NULL. Граф выборки записей будет иметь вид:

Запрос:

```
SELECT * FROM 'Auto'
FULL OUTER JOIN 'Selling' ON 'Auto'.id = 'Selling'.id
```

*Ключевое слово OUTER можно опустить.

id	name	id	sum
1	bmw	1	250
2	opel	NULL	NULL
3	kia	3	300
4	audi	NULL	NULL
NULL	NULL	5	450
NULL	NULL	6	400

То есть мы получили все записи, которые есть в обоих таблицах. Записи у которых значение поля id совпадает соединяются, а у записей для которых совпадений не найдено недостающие поля заполняются значением NULL.

Оператор LEFT JOIN

Левое внешнее объединение (**LEFT OUTER JOIN**). В этом случае получаем все записи удовлетворяющие условию объединения, плюс все оставшиеся записи из внешней таблицы, которые не удовлетворяют условию объединения.
Граф выборки:

Запрос:

```
SELECT * FROM 'Auto'
LEFT OUTER JOIN 'Selling' ON 'Auto'.id = 'Selling'.id
```

id	name	id	sum
1	bmw	1	250
2	opel	NULL	NULL
3	kia	3	300
4	audi	NULL	NULL

*Запрос также можно писать без ключевого слова OUTER.

В итоге здесь мы получили все записи таблицы Auto. Записи для которых были найдены совпадения по полю id в таблице Selling соединяются, для остальных недостающие поля заполняются значением NULL.

Оператор LEFT (RIGHT) JOIN

Левое внешнее объединение (**LEFT OUTER JOIN**). В этом случае получаем все записи удовлетворяющие условию объединения, плюс все оставшиеся записи из внешней таблицы, которые не удовлетворяют условию объединения.
Граф выборки:

Запрос:

```
SELECT * FROM 'Auto'
LEFT OUTER JOIN 'Selling' ON 'Auto'.id = 'Selling'.id
```

*Запрос также можно писать без ключевого слова OUTER.

id	name	id	sum
1	bmw	1	250
2	opel	NULL	NULL
3	kia	3	300
4	audi	NULL	NULL

В итоге здесь мы получили все записи таблицы Auto. Записи для которых были найдены совпадения по полю id в таблице Selling соединяются, для остальных недостающие поля заполняются значением NULL.

Еще существует **правое внешнее объединение** (**RIGHT OUTER JOIN**). Оно работает точно также как и левое объединение, только в качестве внешней таблицы будет использоваться правая (в нашем случае таблица Selling или таблица Б на графе).