**Запросы на выборку и модификацию данных**

Нужно понимать, что все **SQL запросы делятся на четыре группы:**

1. Инструкции или операторы для определения данных(Data Definition Language, DDL).
2. Для манипуляции данными(Data Manipulation Language, DML).
3. Для определения параметров доступа к данным(Data Control Language, DCL).
4. Для управления транзакциями(Transaction Control Language, TCL).

Определение данных подразумевает создание, редактирование и удаление различных объектов базы данных, таких как таблицы (tables), табличные представления (views), синонимы (synonyms), хранимые процедуры, профили пользователя и т.п. **Определение параметров доступа к данным** – это процесс награждения или лишения объектов базы данных различного рода разрешениями, привилегиями и полномочиями, например, предоставление конкретному пользователю базы данных (имеется в виду объект типа user/schema, который определяет права доступа к разделам базы данных в распределенных СУБД, например, в Oracle) возможности осуществлять запросы к конкретной таблице. Управление транзакциями, в самом простом варианте, сводится к возможности сохранить текущие изменения, накопившиеся в результате выполнения последовательности запросов манипуляции данными или целиком их все отменить.

Чаще всего, под SQL запросами понимается именно группа операторов манипуляции данными. Каждая отдельно взятая СУБД поддерживает ту или иную группу SQL запросов в разной мере/объеме, но, наибольшим образом, все они пересекаются именно в реализации операций манипуляции данными. По этой причине будет рассматриваться только эта группа команд: выбор, обновление, добавление и удаление записей из таблиц.

Все запросы на получение практически любых данных из одной или нескольких таблиц выполняются с помощью единственного предложения SELECT.

**В синтаксических конструкциях для обращения к БД используются следующие обозначения:**

* **звездочка (\*) для обозначения "все"**употребляется в обычном для программирования смысле, т.е. "все случаи, удовлетворяющие определению";
* **квадратные скобки ([])** означают, что конструкции, заключенные в эти скобки, являются необязательными (т.е. могут быть опущены);
* **фигурные скобки ({})** означают, что конструкции, заключенные в эти скобки, должны рассматриваться как целые синтаксические единицы, т.е. они позволяют уточнить порядок разбора синтаксических конструкций, заменяя обычные скобки, используемые в синтаксисе SQL;
* **многоточие (…)** указывает на то, что непосредственно предшествующая ему синтаксическая единица факультативно может повторяться один или более раз;
* **прямая черта (|)** означает наличие выбора из двух или более возможностей. Например, обозначение ASC|DESC указывает, можно выбрать один из терминов ASC или DESC; когда же один из элементов выбора заключен в квадратные скобки, то это означает, что он выбирается по умолчанию (так, [ASC]|DESC означает, что отсутствие всей этой конструкции будет восприниматься как выбор ASC);
* **точка с запятой (;)** - завершающий элемент предложений SQL;
* **запятая (,)** используется для разделения элементов списков;
* **пробелы ( )** могут вводиться для повышения наглядности между любыми синтаксическими конструкциями предложений SQL;
* **жирные прописные латинские буквы и символы** используются для написания конструкций языка SQL и должны (если это специально не оговорено) записываться в точности так, как показано;
* **строчные буквы** используются для написания конструкций, которые должны заменяться конкретными значениями, выбранными пользователем, причем, для определенности отдельные слова этих конструкций связываются между собой символом подчеркивания (\_);
* **термины "таблица" и "столбец"** заменяют (с целью сокращения текста синтаксических конструкций) термины "имя\_таблицы", "имя\_столбца", …, соответственно;
* **термин "таблица"**- используется для обобщения таких видов таблиц, как базовая\_таблица, представление или псевдоним; здесь псевдоним служит для временного (на момент выполнения запроса) переименования и (или) создания рабочей копии базовой\_таблицы (представления).

**Оператор SELECT** осуществляет выборку из базы данных и имеет наиболее сложную структуру среди всех операторов языка SQL. Простейший оператор SELECT выглядит так: SELECT \* FROM PC. Он осуществляет выборку всех записей из объекта БД табличного типа с именем PC. При этом, столбцы и строки результирующего набора не упорядочены. Чтобы упорядочить поля результирующего набора, их следует перечислить через запятую в нужном порядке после слова SELECT:

**SELECT price, speed, hd, ram, cd, model, code**

**FROM PC.**

В таблице 1 приводится результат выполнения этого запроса.

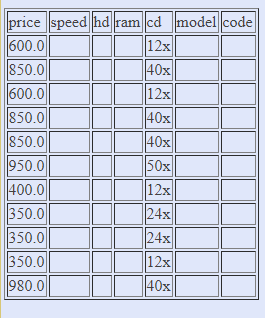


Таблица 1. – Запрос SELECT

Вертикальную проекцию таблицы РC можно получить, если перечислить только необходимые поля. Например, чтобы получить информацию только о скорости процессора и объеме оперативной памяти компьютеров, следует выполнить запрос:

**SELECT speed, ram FROM PC;**

который вернет следующие данные (Таблица 2.):

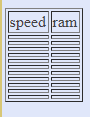


Таблица 2. – Запрос SELECT speed.

Следует отметить, что вертикальная выборка может содержать дубликаты строк в том случае, если она не содержит потенциального ключа, однозначно определяющего запись. В таблице PC потенциальным ключом является поле code, которое выбрано в качестве первичного ключа таблицы. Поскольку это поле отсутствует в запросе, в приведенном выше результирующем наборе имеются дубликаты строк (например, строки 1 и 3). Если требуется получить уникальные строки (скажем, нас интересуют только различные комбинации скорости процессора и объема памяти, а не характеристики всех имеющихся компьютеров), то можно использовать ключевое слово DISTINCT:

**SELECT DISTINCT speed, ram FROM PC;**

что даст такой результат (Таблица 3.):



Таблица 3. – 1-й результат запроса SELECT DISTINCT speed.

Помимо DISTINCT, может применяться также ключевое слово ALL (все строки), которое принимается по умолчанию. Чтобы упорядочить строки результирующего набора, можно выполнить сортировку по любому количеству полей, указанных в предложении SELECT. Для этого используется предложение ORDER BY, являющееся всегда последним предложением в операторе SELECT. При этом, в списке полей могут указываться как имена полей, так и их порядковые позиции в списке предложения SELECT. Так, если требуется упорядочить результирующий набор по объему оперативной памяти в порядке убывания, можно записать:

**SELECT DISTINCT speed, ram**

**FROM PC**

**ORDER BY ram DESC**

Или

**SELECT DISTINCT speed, ram**

**FROM PC**

**ORDER BY 2 DESC**

Результат, приведенный ниже, будет одним и тем же.



Таблица 4. – 2-й результат запроса SELECT DISTINCT speed.

Сортировку можно проводить по возрастанию (параметр ASC принимается по умолчанию) или по убыванию (параметр DESC). Сортировка по двум полям:

**SELECT DISTINCT speed, ram**

**FROM PC**

**ORDER BY ram DESC, speed DESC**

даст следующий результат (Таблица 5.):

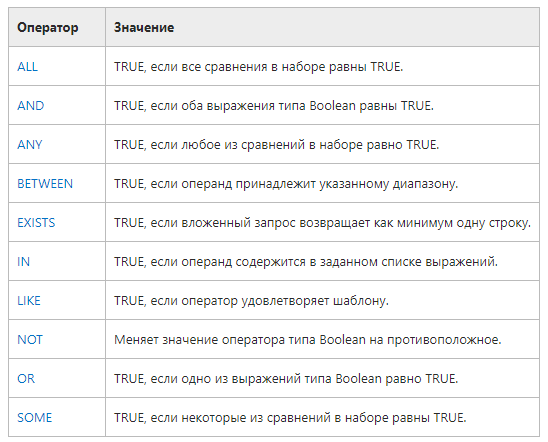


Таблица 5. – 3-й результат запроса SELECT DISTINCT speed.

Горизонтальную выборку реализует предложение WHERE , которое записывается после предложения FROM. При этом, в результирующий набор попадут только те строки из источника записей, для каждой из которых значение предиката равно TRUE. То есть предикат проверяется для каждой записи.

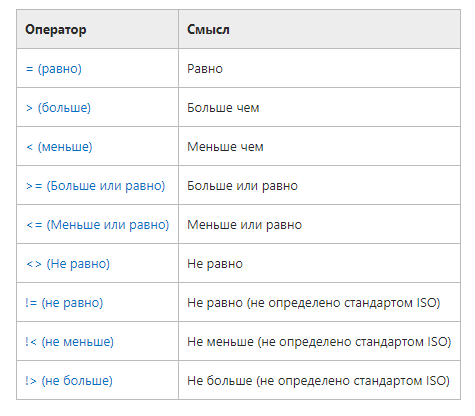
**Логические операторы (Transact-SQL)**

Логические операторы проверяют истину некоторого условия. Логические операторы, например, оператор сравнения, возвращают значение типа Boolean: TRUE, FALSE или UNKNOWN.



**Операторы сравнения (Transact-SQL)**

Операторы сравнения позволяют проверить, одинаковы ли два выражения. Операторы сравнения можно применять ко всем выражениям, за исключением выражений типов text, ntext и image. Операторы сравнения Transact-SQL приведены в следующей таблице:



**Тип данных Boolean**

Результат выполнения оператора сравнения имеет тип данных Boolean. Он может иметь одно из трех значений: TRUE, FALSE и UNKNOWN. Выражения, возвращающие значения типа Boolean, называются «логическими выражениями». В отличие от других типов данных SQL Server, тип Boolean не может быть типом столбца или переменной и не может быть возвращен в результирующем наборе.

**Примеры использования предикатов**

Предикаты представляют собой выражения, принимающие истинностное значение. Они могут представлять собой как одно выражение, так и любую комбинацию из неограниченного количества выражений, построенную с помощью булевых операторов AND, OR или NOT. Кроме того, в этих комбинациях может использоваться SQL-оператор IS, а также круглые скобки для конкретизации порядка выполнения операций

Предикат в языке SQL может принимать одно из трех значений TRUE (истина), FALSE (ложь) или UNKNOWN (неизвестно). Исключение составляют следующие предикаты: NULL (отсутствие значения), EXISTS (существование), UNIQUE (уникальность) и MATCH (совпадение), которые не могут принимать значение UNKNOWN.

Правила комбинирования всех трех истинностных значений легче запомнить, обозначив TRUE как 1, FALSE как 0 и UNKNOWN как 1/2 (где то между истинным и ложным).

**AND** с двумя истинностными значениями дает минимум этих значений. Например, TRUE AND UNKNOWN будет равно UNKNOWN.

**OR** с двумя истинностными значениями дает максимум этих значений. Например, FALSE OR UNKNOWN будет равно UNKNOWN.

Отрицание истинностного значения равно 1 минус данное истинностное значение. Например, NOT UNKNOWN будет равно UNKNOWN.

Помимо этого, используются предикаты сравнения.

Предикат сравнения представляет собой два выражения, соединяемых оператором сравнения. Имеется шесть традиционных операторов сравнения: =, >, <, >=, <=, <>.

Данные типа NUMERIC (числа) сравниваются в соответствии с их алгебраическим значением.

Данные типа CHARACTER STRING (символьные строки) сравниваются в соответствии с их алфавитной последовательностью. Если a1a2…an и b1b2…bn - две последовательности символов, то первая "меньше" второй, если а1<b1, или а1=b1 и а2<b2 и т.д. Считается также, что а1а2…аn<b1b2…bm, если n<m и а1а2…аn=b1b2…bn, т.е. если первая строка является префиксом второй. Например, 'folder'<'for', т.к. первые две буквы этих строк совпадают, а третья буква строки 'folder' предшествует третьей букве строки 'for'. Также справедливо неравенство 'bar' < 'barber', поскольку первая строка является префиксом второй.

Данные типа **DATETIME (дата/время)**сравниваются в хронологическом порядке.

Данные типа I**NTERVAL (временной интервал)**преобразуются в соответствующие типы, а затем сравниваются как обычные числовые значения типа NUMERIC.

**Пример.** Получить информацию о компьютерах, имеющих частоту процессора не менее 500 Мгц и цену ниже $800:

**SELECT \* FROM PC**

**WHERE speed >= 500 AND price < 800;**

Запрос возвращает следующие данные (Таблица 6.):

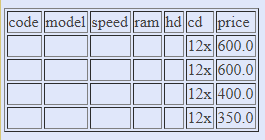


Таблица 6. – Пример информационного запроса.

Существуют так же и другие предикаты, например, BETWEEN, IN, LIKE.

Имена столбцов, указанных в предложении SELECT, можно переименовать. Это делает результаты более читабельными, поскольку имена полей в таблицах часто сокращают с целью упрощения набора. Ключевое слово AS, используемое для переименования, согласно стандарту можно и опустить, т.к. оно неявно подразумевается.

Например, запрос:

**SELECT ram AS Mb, hd Gb**

**FROM PC**

**WHERE cd = '24x';**

переименует столбец ram в Mb (мегабайты), а столбец hd - в Gb (гигабайты). Этот запрос возвратит объемы оперативной памяти и жесткого диска для тех компьютеров, которые имеют 24-скоростной CD-ROM (Таблица 7.):



Таблица 7. – Пример запроса SELECT AS.

**Получение итоговых значений:**

Существует возможность получения итоговых (агрегатных) функций. Стандартом предусмотрены следующие агрегатные функции (Таблица 8.):

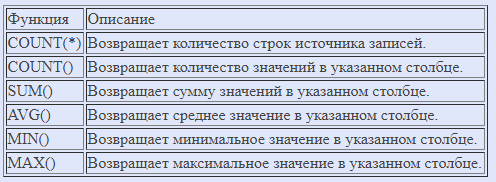


Таблица 8 .– Описание (агрегатных) функций.

Все эти функции возвращают единственное значение. При этом, функции COUNT, MIN и MAX применимы к любым типам данных, в то время как SUM и AVG используются только для числовых полей. Разница между функцией COUNT(\*) и COUNT() состоит в том, что вторая при подсчете не учитывает NULL-значения.

**Пример.** Найти минимальную и максимальную цену на персональные компьютеры:

**SELECT MIN(price) AS Min\_price, MAX(price) AS Max\_price**

**FROM PC;**

Результатом будет единственная строка, содержащая агрегатные значения (таблица 9.):

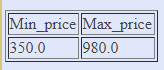


Таблица 9. – Строка содержащая (агрегатные) значения.

Для просмотра данных наиболее удобно использовать совместно значения оператора COUNT - счетчик (позволяет узнать количество записей в запросе), и оператора CURSOR - позволяет принимать не все записи сразу, а по одной (указанной пользователем).

**Иные операторы**

Язык манипуляции данными (DML - Data Manipulation Language) помимо оператора SELECT, осуществляющего извлечение информации из базы данных, включает операторы, изменяющие состояние данных. Этими операторами являются:

* **INSERT**- добавление записей (строк) в таблицу БД.
* **UPDATE**- обновление данных в столбце таблицы БД.
* **DELETE**- удаление записей из таблицы БД.

**Оператор INSERT.**

Оператор INSERT вставляет новые строки в таблицу. При этом, значения столбцов могут представлять собой литеральные константы либо являться результатом выполнения подзапроса. В первом случае для вставки каждой строки используется отдельный оператор INSERT; во втором случае - будет вставлено столько строк, сколько возвращается подзапросом.

**Синтаксис оператора:**

**INSERT INTO <имя таблицы>[(<имя столбца>,...)]**

**{VALUES (< значение столбца>,…)}**

**| <выражение запроса>**

**| {DEFAULT VALUES};**

Как видно из представленного синтаксиса, список столбцов не является обязательным. В том случае, если он отсутствует, список вставляемых значений должен быть полный, т.е. обеспечивать значения для всех столбцов таблицы. При этом порядок значений должен соответствовать порядку столбцов, заданному оператором CREATE TABLE для таблицы, в которую вставляются строки. Кроме того, каждое из этих значений должно быть того же типа (или приводиться к нему), что и тип, определенный для соответствующего столбца в операторе CREATE TABLE.

В качестве примера рассмотрим вставку строки в таблицу Product, созданную следующим оператором CREATE TABLE:

**CREATE TABLE [dbo].[product] (**

**[maker] [char] (1) NOT NULL ,**

**[model] [varchar] (4) NOT NULL ,**

**[type] [varchar] (7) NOT NULL )**

Пусть требуется добавить в эту таблицу модель ПК 1157 производителя B. Это можно сделать следующим оператором:

**INSERT INTO Product VALUES ('B', 1157, 'PC');**

Если задать список столбцов, то можно изменить "естественный" порядок их следования:

**INSERT INTO Product (type, model, maker) VALUES ('PC', 1157, 'B');**

Казалось бы, это совершенно излишняя возможность, которая делает конструкцию только более громоздкой. Однако она становится выигрышной, если столбцы имеют значения по умолчанию. Рассмотрим следующую структуру таблицы:

**CREATE TABLE [product\_D] (**

**[maker] [char] (1) NULL ,**

**[model] [varchar] (4) NULL ,**

**[type] [varchar] (7) NOT NULL DEFAULT 'PC' )**

Отметим, что здесь значения всех столбцов имеют значения по умолчанию (первые два - NULL, а последний столбец - type - 'PC'). Теперь мы могли бы написать:

**INSERT INTO Product\_D (model, maker) VALUES (1157, 'B');**

В этом случае, отсутствующее значение при вставке строки будет заменено значением по умолчанию - 'PC'. Заметим, что если для столбца в операторе CREATE TABLE не указано значение по умолчанию и не указано ограничение NOT NULL, запрещающее использование NULL в данном столбце таблицы, то подразумевается значение по умолчанию NULL.

Возникает вопрос: а можно ли не указывать список столбцов и, тем не менее, воспользоваться значениями по умолчанию? Ответ положительный. Для этого нужно вместо явного указания значения использовать зарезервированное слово DEFAULT:

**INSERT INTO Product\_D VALUES ('B', 1158, DEFAULT);**

Поскольку все столбцы имеют значения по умолчанию, для вставки строки со значениями по умолчанию можно было бы написать:

**INSERT INTO Product\_D VALUES (DEFAULT, DEFAULT, DEFAULT);**

Однако для этого случая предназначена специальная конструкция DEFAULT VALUES (смотри синтаксис оператора), с помощью которой вышеприведенный оператор можно переписать в виде

**INSERT INTO Product\_D DEFAULT VALUES;**

Заметим, что, при вставке строки в таблицу, проверяются все ограничения, наложенные на данную таблицу. Это могут быть ограничения первичного ключа или уникального индекса, проверочные ограничения типа CHECK, ограничения ссылочной целостности. В случае нарушения какого-либо ограничения вставка строки будет отвергнута.

Рассмотрим теперь случай использования подзапроса. Пусть нам требуется вставить в таблицу Product\_D все строки из таблицы Product, относящиеся к моделям персональных компьютеров (type = 'PC'). Поскольку необходимые нам значения уже имеются в некоторой таблице, то формирование вставляемых строк вручную, во-первых, является неэффективным, а, во-вторых, может допускать ошибки ввода. Использование подзапроса решает эти проблемы

**INSERT INTO Product\_D SELECT \* FROM Product WHERE type = 'PC';**

Использование в подзапросе символа "\*" является в данном случае оправданным, т.к. порядок следования столбцов является одинаковым для обеих таблиц. Если бы это было не так, следовало бы использовать список столбцов либо в операторе INSERT, либо в подзапросе, либо в обоих местах, который приводил бы в соответствие порядок следования столбцов.

Преодолеть ограничение на вставку одной строки в операторе INSERT, при использовании VALUES, позволяет искусственный прием использования подзапроса, формирующего строку с предложением UNION ALL. Если нам требуется вставить несколько строк при помощи одного оператора INSERT, можно написать:

**INSERT INTO Product\_D**

**SELECT 'B' AS maker, 1158 AS model, 'PC' AS type**

**UNION ALL**

**SELECT 'C', 2190, 'Laptop'**

**UNION ALL**

**SELECT 'D', 3219, 'Printer';**

Использование UNION ALL предпочтительней UNION даже если гарантировано отсутствие строк-дубликатов, т.к. в этом случае не будет выполняться проверка для исключения дубликатов.

* **Оператор UPDATE**

Оператор UPDATE изменяет имеющиеся данные в таблице. Команда имеет следующий синтаксис:

**UPDATE**

**SET {имя столбца = {выражение для вычисления значения столбца**

**| NULL**

**| DEFAULT},...}**

**[ {WHERE }];**

С помощью одного оператора могут быть заданы значения для любого количества столбцов. Однако в одном и том же операторе UPDATE можно вносить изменения в каждый столбец указанной таблицы только один раз. При отсутствии предложения WHERE будут обновлены все строки таблицы.

Если столбец допускает NULL-значение, то его можно указать в явном виде. Кроме того, можно заменить имеющееся значение на значение по умолчанию (DEFAULT) для данного столбца.

Ссылка на "выражение" может относиться к текущим значениям в изменяемой таблице. Например, мы можем уменьшить все цены ПК-блокнотов на 10 процентов с помощью следующего оператора:

**UPDATE Laptop SET price=price\*0.9**

Разрешается также значения одних столбцов присваивать другим столбцам. Пусть, например, требуется заменить жесткие диски менее 10 Гб в ПК-блокнотах. При этом емкость новых дисков должна составлять половину объема RAM, имеющейся в данных устройствах. Эту задачу можно решить следующим образом:

**UPDATE Laptop SET hd=ram/2 WHERE hd**

Естественно, типы данных столбцов hd и ram должны быть совместимы. Для приведения типов может использоваться выражение cast.

Если требуется изменять данные, в зависимости от содержимого некоторого столбца, можно воспользоваться выражением case.

Если, скажем, нужно поставить жесткие диски объемом 20 Гб на ПК-блокноты с памятью менее 128 Мб и 40 гигабайтные - на остальные ПК-блокноты, то можно написать такой запрос:

**UPDATE Laptop**

**SET hd = CASE WHEN ram<128 THEN 20 ELSE 40 END**

Для вычисления значений столбцов допускается также использование подзапросов. Например, требуется укомплектовать все ПК-блокноты самыми быстрыми процессорами из имеющихся. Тогда можно написать:

**UPDATE Laptop**

**SET speed = (SELECT MAX(speed) FROM Laptop)**

Необходимо сказать несколько слов об автоинкрементируемых столбцах. Если столбец code в таблице Laptop определен как IDENTITY(1,1), то поступают следующим образом. Сначала необходимо вставить нужную строку, используя SET IDENTITY\_INSERT, после чего удалить старую строку:

**SET IDENTITY\_INSERT Laptop ON**

**INSERT INTO Laptop\_ID(code, model, speed, ram, hd, price, screen)**

**SELECT 5, model, speed, ram, hd, price, screen**

**FROM Laptop\_ID WHERE code=4**

**DELETE FROM Laptop\_ID WHERE code=4**

Разумеется, другой строки со значением code=5 в таблице быть не должно.

В Transact-SQL оператор UPDATE расширяет стандарт за счет использования необязательного предложения FROM. В этом предложении специфицируется таблица, обеспечивающая критерий для операции обновления. Дополнительную гибкость здесь дает использование операций соединения таблиц.

* **Оператор DELETE**

Пример. Требуется удалить из таблицы Laptop все ПК-блокноты с размером экрана менее 12 дюймов.

**DELETE FROM Laptop**

**WHERE screen**

Все блокноты можно удалить с помощью оператора DELETE FROM Laptop или TRUNCATE TABLE Laptop.

Transact-SQL расширяет синтаксис оператора DELETE, вводя дополнительное предложение FROM.

При помощи источника табличного типа можно конкретизировать данные, удаляемые из таблицы в первом предложении FROM.

При помощи этого предложения можно выполнять соединения таблиц, что логически заменяет использование подзапросов в предложении WHERE для идентификации удаляемых строк.

Поясним сказанное на примере. Пусть требуется удалить те модели ПК из таблицы Product, для которых нет соответствующих строк в таблице PC.

Используя стандартный синтаксис, эту задачу можно решить следующим запросом:

**DELETE FROM Product**

**WHERE type='pc' AND model NOT IN (SELECT model FROM PC)**

Заметим, что предикат type='pc' необходим здесь, чтобы не были удалены также модели принтеров и ПК-блокнотов.

Эту же задачу можно решить с помощью дополнительного предложения FROM следующим образом:

**DELETE FROM Product**

**FROM Product pr LEFT JOIN PC ON pr.model=pc.model**

**WHERE type='pc' AND pc.model IS NULL**

Здесь используется внешнее соединение, в результате чего столбец pc.model для моделей ПК, отсутствующих в таблице PC, будет содержать NULL-значение, что и используется для идентификации подлежащих удалению строк.

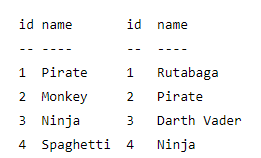
**Виды JOIN:**

* **(INNER) JOIN**возвращает записи, имеющие соответствующие значения в обеих таблицах.
* **LEFT (OUTER) JOIN** возвращает все записи из левой таблицы и соответствующие записи из правой таблицы.
* **RIGHT (OUTER) JOIN** возвращает все записи из правой таблицы и сопоставленные записи из левой таблицы.
* **FULL (OUTER) JOIN** возвращает все записи, когда есть совпадение в левой или правой таблице.
* **CROSS JOIN** создает набор результатов, который представляет собой количество строк в первой таблице, умноженное на количество строк во второй таблице.

**Использование JOIN**

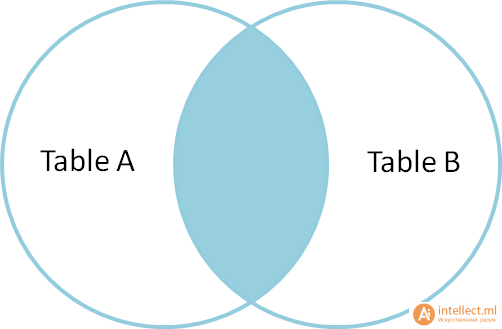
Использование синтаксиса SQL JOINS при работе с базами данных достаточно популярно, без них не обходится любой серьезный SQL запрос.

Предположим, что у нас есть две следующие таблицы. Слева Таблица A, и таблица B справа. Поместим в каждой из них по 4 записи (строки).



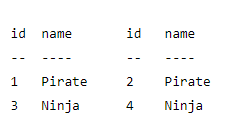
Давайте соединим эти таблицы с помощью SQL join по столбцу "name" несколькими способами и посмотрим, как это будет выглядеть на диаграммах Венна.

**Inner join(внутреннее присоединение)**производит выбор только строк, которые есть и в таблице А, и в таблице В.

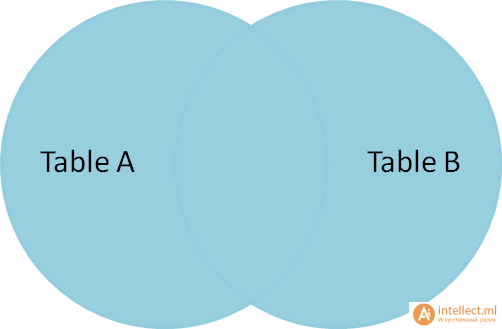


**SELECT \* FROM TableA**

**INNER JOINTableB ON TableA.name = TableB.name**

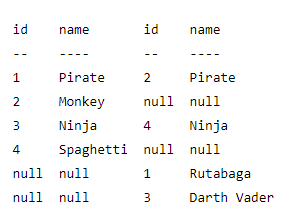


**Full outer join (полное внешнее соединение - объединение)** производит выбор всех строк из таблиц А и В, причем со всеми возможными вариантами. Если с какой-либо стороны не будет записи, то недостающая запись будет содержать пустую строку (null значения).

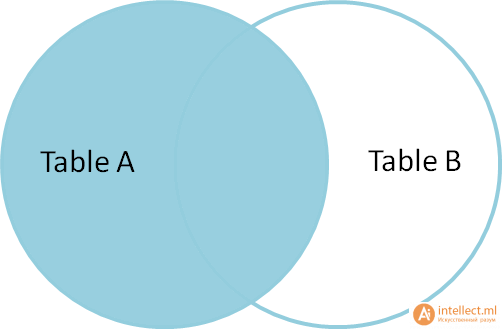


**SELECT \* FROM TableA**

**FULL OUTER JOIN TableB ON TableA.name = TableB.name**

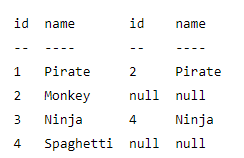


**Left outer join(левое внешнее соединение)** производит выбор всех строк таблицы А с доступными строками таблицы В. Если строки таблицы В не найдены, то подставляется пустой результат (null).

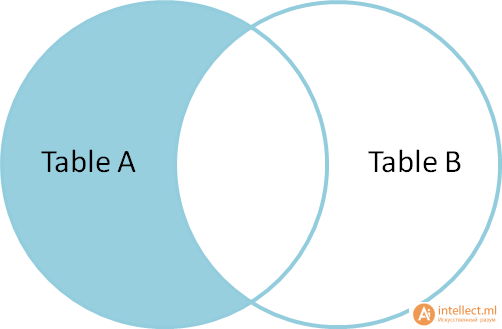


**SELECT \* FROM TableA**

**LEFT OUTER JOINTableB ON TableA.name = TableB.name**



Чтобы произвести выбор строк из Таблицы A, которых нет в Таблице Б, мы выполняем тот же самый LEFT OUTER JOIN, затем исключаем строки, которые заполнены в Таблице Б. То есть выбрать все записи таблицы А, которых нет в Таблице В, мы выполняем тоже jeft outer join, но исключаем пустые записи таблицы В.

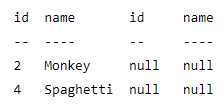


С добавлением условия получаем:

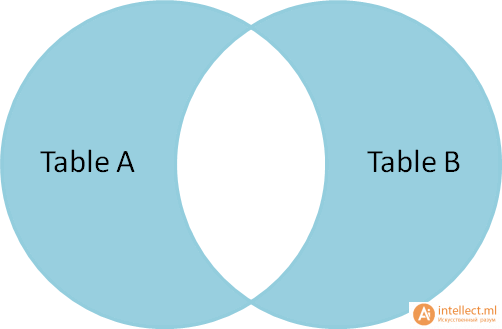
**SELECT \* FROM TableA**

**LEFT OUTER JOINTableB ON TableA.name = TableB.name**

**WHERE TableB.id IS null**



Чтобы выбрать уникальные записи таблиц А и В, мы выполняем FULL OUTER JOIN, а затем исключаем записи, которые принадлежат и таблице А, и таблице Б с помощью условия WHERE.



**SELECT \* FROM TableA**

**FULL OUTER JOINTableB ON TableA.name = TableB.name**

**WHERE TableA.id IS null OR TableB.id IS null**

