Общая характеристика структурированного языка запросов sql

SQL (англ. structured query language — «язык структурированных запросов») — формальный непроцедурный язык программирования, применяемый для создания, модификации и управления данными в произвольной реляционной базе данных, управляемой соответствующей системой управления базами данных (СУБД). SQL основывается на исчислении кортежей.

SQL является прежде всего информационно-логическим языком, предназначенным для описания, изменения и извлечения данных, хранимых в реляционных базах данных. SQL можно назвать языком программирования, при этом он не является тьюринг-полным, но вместе с тем стандарт языка спецификацией SQL/PSM предусматривает возможность его процедурных расширений.

Изначально SQL был основным способом работы пользователя с базой данных и позволял выполнять следующий набор операций:

− создание в базе данных новой таблицы;

− добавление в таблицу новых записей;

− изменение записей;

− удаление записей;

− выборка записей из одной или нескольких таблиц (в соответствии с заданным условием);

− изменение структур таблиц.

Со временем SQL усложнился — обогатился новыми конструкциями, обеспечил возможность описания и управления новыми хранимыми объектами (например, индексы, представления, триггеры и хранимые процедуры) — и стал приобретать черты, свойственные языкам программирования.

При всех своих изменениях SQL остаётся единственным механизмом связи между прикладным программным обеспечением и базой данных. В то же время современные СУБД, а также информационные системы, использующие СУБД, предоставляют пользователю развитые средства визуального построения запросов.

Каждое предложение SQL — это либо запрос данных из базы, либо обращение к базе данных, которое приводит к изменению данных в базе. В соответствии с тем, какие изменения происходят в базе данных, различают следующие типы запросов:

− запросы на создание или изменение в базе данных новых или существующих объектов (при этом в запросе описывается тип и структура создаваемого или изменяемого объекта);

− запросы на получение данных;

− запросы на добавление новых данных (записей);

− запросы на удаление данных;

− обращения к СУБД.

Основным объектом хранения реляционной базы данных является таблица, поэтому все SQL-запросы — это операции над таблицами. В соответствии с этим, запросы делятся на:

− запросы, оперирующие самими таблицами (создание и изменение таблиц);

− запросы, оперирующие с отдельными записями (или строками таблиц) или наборами записей.

Каждая таблица описывается в виде перечисления своих полей (столбцов таблицы) с указанием

− типа хранимых в каждом поле значений;

− связей между таблицами (задание первичных и вторичных ключей);

− информации, необходимой для построения индексов.

Запросы первого типа в свою очередь делятся на запросы, предназначенные для создания в базе данных новых таблиц, и на запросы, предназначенные для изменения уже существующих таблиц. Запросы второго типа оперируют со строками, и их можно разделить на запросы следующего вида:

− вставка новой строки;

− изменение значений полей строки или набора строк;

− удаление строки или набора строк.

Самый главный вид запроса — это запрос, возвращающий (пользователю) некоторый набор строк, с которым можно осуществить одну из трёх операций:

− просмотреть полученный набор;

− изменить все записи набора;

− удалить все записи набора.

Таким образом, использование SQL сводится, по сути, к формированию всевозможных выборок строк и совершению операций над всеми записями, входящими в набор.

Первые разработки

В начале 1970-х годов в одной из исследовательских лабораторий компании IBM была разработана экспериментальная реляционная СУБД IBM System R, для которой затем был создан специальный язык SEQUEL, позволявший относительно просто управлять данными в этой СУБД. Аббревиатура SEQUEL расшифровывалась как Structured English QUEry Language — «структурированный английский язык запросов». Позже по юридическим соображениям[2] язык SEQUEL был переименован в SQL. Когда в 1986 году первый стандарт языка SQL был принят ANSI (American National Standards Institute), официальным произношением стало [ˌes kjuː' el] — эс-кью-эл. Несмотря на это, даже англоязычные специалисты зачастую

продолжают читать SQL как сиквел (по-русски часто говорят «эс-ку-эль» или используют жаргонизм «скуль»).

Целью разработки было создание простого непроцедурного языка, которым мог воспользоваться любой пользователь, даже не имеющий навыков программирования. Собственно разработкой языка запросов занимались Дональд Чэмбэрлин (Donald D. Chamberlin) и Рэй Бойс (Ray Boyce). Пэт Селинджер (Pat Selinger) занималась разработкой стоимостного оптимизатора (cost-based optimizer), Рэймонд Лори (Raymond Lorie) занимался компилятором запросов.

Стоит отметить, что SEQUEL был не единственным языком подобного назначения. В Калифорнийском Университете Беркли была разработана некоммерческая СУБД Ingres (являвшаяся, между прочим, дальним прародителем популярной сейчас некоммерческой СУБД PostgreSQL), которая являлась реляционной СУБД, но использовала свой собственный язык QUEL, который, однако, не выдержал конкуренции по количеству поддерживающих его СУБД по сравнению с языком SQL.

Первыми СУБД, поддерживающими новый язык, стали в 1979 году Oracle V2 для машин VAX от компании Relational Software Inc. (впоследствии ставшей компанией Oracle) и System/38 от IBM, основанная на System/R.

Стандартизация

Поскольку к началу 1980-х годов существовало несколько вариантов СУБД от разных производителей, причём каждый из них обладал собственной реализацией языка запросов, было принято решение разработать стандарт языка, который будет гарантировать переносимость ПО с одной СУБД на другую (при условии, что они будут поддерживать этот стандарт).

В 1983 году Международная организация по стандартизации (ISO) и Американский национальный институт стандартов (ANSI) приступили к разработке стандарта языка SQL. По прошествии множества консультаций и отклонения нескольких предварительных вариантов в 1986 году ANSI представил свою первую версию стандарта, описанного в документе ANSI X3.135-1986 под названием «Database Language SQL» (Язык баз данных SQL). Неофициально этот стандарт SQL-86 получил название SQL1. Год спустя, была завершена работа над версией стандарта ISO 9075-1987 под тем же названием. Разработка этого стандарта велась под патронажем Технического Комитета TC97 (англ. Technical Committee TC97), областью деятельности которого являлись процессы вычисления и обработки информации (англ. Computing and Information Processing). Именно его подразделение, именуемое как Подкомитет SC21 (англ. Subcommittee SC21) курировало разработку стандарта, что стало залогом идентичности стандартов ISO и ANSI для SQL1 (SQL-86).

Стандарт SQL1 разделялся на два уровня. Первый уровень представлял собой подмножество второго уровня, описывавшего весь документ в целом.

То есть, такая структура предусматривала, что не все спецификации стандарта SQL1 будут относиться к Уровню 1. Тем самым, поставщик, заявлявший о поддержке данного стандарта, должен был заявлять об уровне, которому соответствует его реализация языка SQL. Это значительно облегчило принятие и поддержку стандарта, поскольку производители могли реализовывать его поддержку в два этапа.

Со временем к стандарту накопилось несколько замечаний и пожеланий, особенно с точки зрения обеспечения целостности и корректности данных, в результате чего в 1989 году данный стандарт был расширен, получив название SQL89. В частности, в него была добавлена концепция первичного и внешнего ключей. ISO-версия документа получила название ISO 9075:1989 «Database Language SQL with Integrity Enhancements» (Язык баз данных SQL с добавлением контроля целостности). параллельно была закончена и ANSI-версия.

Сразу после завершения работы над стандартом SQL1 в 1987 году была начата работа над новой версией стандарта, который должен был заменить стандарт SQL89, получив название SQL2, поскольку дата принятия документа на тот момент была неизвестна. Таким образом, фактически SQL89 и SQL2 разрабатывались параллельно. Новая версия стандарта была принята в 1992 году, заменив стандарт SQL89. Новый стандарт, озаглавленный как SQL92, представлял собой по сути расширение стандарта SQL1, включив в себя множество дополнений имевшихся в предыдущих версиях инструкций.

Как и SQL1, SQL92 также был разделён на несколько уровней, однако, во-впервых, число уровней было увеличено с двух до трёх, а во-вторых они получили названия вместо порядковых цифр: начальный (англ. entry), средний (англ. intermediate), полный (англ. full). Уровень «полный» как и Уровень 2 в SQL1 подразумевал весь стандарт целиком. Уровень «начальный» представлял собой подмножество уровня «средний», в свою очередь представлявшего собой подмножество уровня «полный». Уровень «начальный» был сравним с Уровнем 2 стандарта SQL1, но спецификации этого уровня были несколько расширены. Таким образом, цепочка включений уровней стандартов выглядела примерно следующим образом: SQL1 Уровень 1 → SQL1 Уровень 2 → SQL92 «Начальный» → SQL92 «Средний» → SQL92 «Полный».

После принятия стандарта SQL92 к нему были добавлены ещё несколько документов, расширявших функциональность языка. так, в 1995 году был принят стандарт SQL/CLI (Call Level Interface, интерфейс уровня вызовов), впоследствии переименованный в CLI95. На следующий год был принят стандарт SQL/PSM (Persistent Stored Modules, постоянно хранимые модули), получивший название PSM-96.[3]

Следующим стандартом стал SQL:1999 (SQL3). В настоящее время действует стандарт, принятый в 2003 году (SQL:2003) с небольшими модификациями, внесёнными позже (SQL:2008). История версий стандарта:

Год	Название	Иное название	Изменения
	SQL-86	SQL-87	Первый вариант стандарта, принятый институтом ANSI и одобренный ISO в 1987 году.
	SQL-89	FIPS 127-1	Немного доработанный вариант предыдущего стандарта.
	SQL-92	SQL2, FIPS 127-2	Значительные изменения (ISO 9075); уровень *Entry Level* стандарта SQL-92 был принят как стандарт FIPS 127-2.
	SQL:1999	SQL3	Добавлена поддержка регулярных выражений, рекурсивных запросов, поддержка триггеров, базовые процедурные расширения, нескалярные типы данных и некоторые объектно-ориентированные возможности.
	SQL:2003		Введены расширения для работы с XML-данными, оконные функции (применяемые для работы с OLAP-базами данных), генераторы последовательностей и основанные на них типы данных.
	SQL:2006		Функциональность работы с XML-данными значительно расширена. Появилась возможность совместно использовать в запросах SQL и XQuery
	SQL:2008		Улучшены возможности оконных функций, устранены некоторые неоднозначности стандарта SQL:2003

Категории команд SQL

Реализация в SQL концепции операций, ориентированных на табличное представление данных, позволила создать компактный язык с небольшим набором предложений. Язык SQL может использоваться как для выполнения запросов к данным, так и для построения прикладных программ.

Основные категории команд языка SQL предназначены для выполнения различных функций, включая построение объектов базы данных и манипулирование ими, начальную загрузку данных в таблицы, обновление и удаление существующей информации, выполнение запросов к базе данных, управление доступом к ней и ее общее администрирование.

Основные категории команд языка SQL:

− DDL – язык определения данных;
− DML – язык манипулирования данными;
− DQL – язык запросов;
− DCL – язык управления данными;
− команды администрирования данных;
− команды управления транзакциями

Определение структур базы данных (DDL)

Язык определения данных (Data Definition Language, DDL) позволяет создавать и изменять структуру объектов базы данных, например, создавать и удалять таблицы. Основными командами языка DDL являются следующие: CREATE TABLE, ALTER TABLE, DROP TABLE, CREATE INDEX, ALTER INDEX, DROP INDEX.

Манипулирование данными (DML)

Язык манипулирования данными (Data Manipulation Language, DML) используется для манипулирования информацией внутри объектов реляционной базы данных посредством трех основных команд: INSERT, UPDATE, DELETE.

Выборка данных (DQL)

Язык запросов DQL наиболее известен пользователям реляционной базы данных, несмотря на то, что он включает всего одну команду SELECT. Эта команда вместе со своими многочисленными опциями и предложениями используется для формирования запросов к реляционной базе данных.

Язык управления данными (DCL - Data Control Language)

Команды управления данными позволяют управлять доступом к информации, находящейся внутри базы данных. Как правило, они используются для создания объектов, связанных с доступом к данным, а также служат для контроля над распределением привилегий между пользователями. Команды управления данными следующие: GRANT, REVOKE.

Команды администрирования данных

С помощью команд администрирования данных пользователь осуществляет контроль за выполняемыми действиями и анализирует операции базы данных; они также могут оказаться полезными при анализе производительности системы. Не следует путать администрирование данных с администрированием базы данных, которое представляет собой общее управление базой данных и подразумевает использование команд всех уровней.

Команды управления транзакциями

Существуют следующие команды, позволяющие управлять транзакциями базы данных: COMMIT, ROLLBACK, SAVEPOINT, SET TRANSACTION.

Оператор выбора select языка sql
Применение агрегатных функций и вложенных запросов

Запросы на выборку SELECT. Основным оператором языка SQL, позволяющим осуществлять отбор информации из базы данных, является оператор SELECT, который в простейшем виде может быть задан следующим образом:

SELECT <список колонок, включаемых в ответ> FROM <список таблиц> WHERE <условие>.

Предложения SELECT (отобрать) и FROM (из) должны присутствовать обязательно. Условие WHERE (где) может быть опущено. Тогда в ответ войдут все строки, имеющиеся в таблице (если быть точным, то надо иметь в виду, что такой оператор должен соответствовать реляционной операции «проекция», и если в результате проекции на выбранные столбцы будут появляться дублирующие строки, то они должны быть удалены из ответа; однако SQL позволяет управлять выводом в ответ повторяющихся строк, и можно добиться как вывода только уникальных строк, так и включения в ответ повторяющихся строк).

Оператор SELECT может включать в себя и другие предложения, позволяющие, в частности, осуществлять упорядоченность ответа, выполнять обобщающие функции.

Если в ответ должны войти все колонки, имеющиеся в исходной таблице, то вместо их перечисления в SELECT можно поставить знак «*».

Так, например, запрос «Выдать всю информацию из таблицы «Kadr» по сотрудникам, чей возраст (vozr) равен 40 годам» может быть представлен на SQL следующим образом:

SELECT * FROM kadr WHERE vozr=40;

Условие, задаваемое в предложении WHERE, может быть простым и сложным. Для формулирования сложного условия могут быть использованы логические операторы AND и OR. Так, например, запрос «Выдать всю информацию из таблицы «Kadr» по мужчинам, чей возраст равен 40 годам» (т.е. пол=«мужской» И «возраст=40 лет) может быть представлен на SQL следующим образом:

SELECT * FROM kadr WHERE vozr = 40 AND pol = «м»;

Оператор SELECT оперирует над множествами и результатом обработки в общем случае является множество строк. К этим множествам могут быть применены теоретико-множественные операции объединение (UNION), пересечение (INTERSECTION), разность (DIFFERENCE, MINUS, EXCEPT) и др. В разных реализациях языка SQL наборы теоретико-множественных операций различаются.

Язык SQL позволяет запрашивать вычисляемые значения. В этом случае в предложении SELECT указывается выражение для вычисления значения колонки. Например, в приведенном ниже предложении запрашивается вывод стоимости поставленных товаров путем ее вычисления на основе хранящихся в таблице «Postypl» данных о количестве (kolv) поставленных товаров и их цены (cena):

SELECT naimprod, datapost, kolv*cena FROM postypl;

Запрос может быть *простым,* состоящим из одного оператора SELECT, и *вложенным,* когда один оператор SELECT включается в состав другого оператора. Этот включенный оператор называется *подзапросом* (subselect) или *подчиненным запросом.* Существуют два типа вложенных подзапросов: *обычный* и *коррелированный.* В обычном подзапросе внутренний запрос выполняется первым, и его результат используется для выполнения основного запроса. В коррелированном подзапросе внешний запрос выполняется первым, и его результат используется для выполнения внутреннего запроса. Внутренний запрос выполняется для каждой строки, возвращенной внешним запросом.

В запросе можно указать упорядоченность ответа по определенному признаку (полю, совокупности полей, выражению).

Возможна подгруппировка данных в целях получения подытогов или других обобщающих величин (среднее, минимум, максимум и др.). Набор агрегатных функций отличается в разных системах. В запросе допускается только один уровень группировки. Группировка может осуществляться как по одному полю, так и по совокупности нолей.

При выполнении запроса может возникнуть необходимость соединения двух или более таблиц. Возможны разные способы задания условия соединения (вложенные запросы, задание условия соединения в предложении WHERE, операция JOIN в предложении FROM).

Соединение таблиц
Операция JOIN и ее виды

Соединение – это операция, когда таблицы сравниваются между собой построчно и появляется возможность вывода столбцов из всех таблиц, участвующих в соединении.

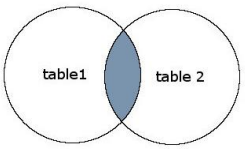
SQL JOINS используются для извлечения данных из нескольких таблиц. SQL JOIN выполняется всякий раз, когда две или более таблицы перечислены в операторе SQL.

Существует 4 различных типа соединений SQL:

- SQL INNER JOIN (иногда называется простым соединением)
- SQL LEFT OUTER JOIN (иногда называется LEFT JOIN)
- SQL RIGHT OUTER JOIN (иногда называется RIGHT JOIN)
- SQL FULL OUTER JOIN (иногда называется FULL JOIN)

Синтаксис INNER JOIN в SQL:

```
SELECT columns
FROM table1
INNER JOIN table2
ON table1.column = table2.column;
```



SQL INNER JOIN будет возвращать записи, где пересекаются table1 и table2.

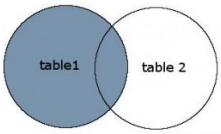
SQL LEFT OUTER JOIN

Другой тип соединения называется LEFT OUTER JOIN. Этот тип соединения возвращает все строки из таблиц с левосторонним соединением, указанным в условии ON, и только те строки из другой таблицы, где объединяемые поля равны (выполняется условие соединения).

Синтаксис для LEFT OUTER JOIN в SQL:

```
SELECT columns
FROM table1
LEFT [OUTER] JOIN table2
ON table1.column = table2.column;
```

В некоторых базах данных ключевое слово OUTER опущено и записывается просто как LEFT JOIN.



SQL LEFT OUTER JOIN возвращает все записи из table1 и только те записи из table2, которые пересекаются с table1.

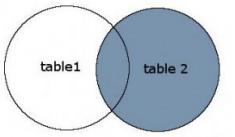
SQL RIGHT OUTER JOIN JOIN

Другой тип соединения называется SQL RIGHT OUTER JOIN. Этот тип соединения возвращает все строки из таблиц с правосторонним соединением, указанным в условии ON, и только те строки из другой таблицы, где объединяемые поля равны (выполняется условие соединения).

Синтаксис для RIGHT OUTER JOIN в SQL:

```
SELECT columns
FROM table1
RIGHT [OUTER] JOIN table2
ON table1.column = table2.column;
```

В некоторых базах данных ключевое слово OUTER опущено и записывается просто как RIGHT JOIN.



SQL RIGHT OUTER JOIN возвращает все записи из table2 и только те записи из table1, которые пересекаются с table2.

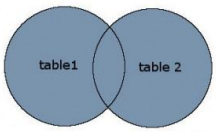
SQL FULL OUTER JOIN

Другой тип объединения называется SQL FULL OUTER JOIN. Этот тип объединения возвращает все строки из LEFT таблицы и RIGHT таблицы со значениями NULL в месте, где условие соединения не выполняется.

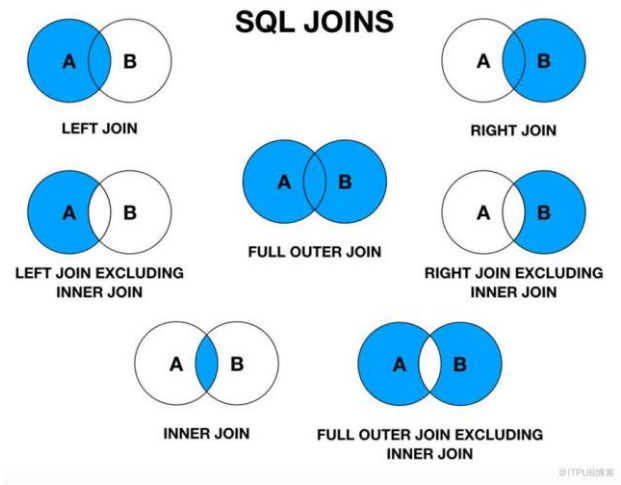
Синтаксис для SQL FULL OUTER JOIN:

```
SELECT columns
FROM table1
FULL [OUTER] JOIN table2
ON table1.column = table2.column;
```

В некоторых базах данных ключевое слово OUTER опускается и записывается просто как FULL JOIN.



SQL FULL OUTER JOIN возвращает все записи из таблиц table1 и table2.



В некоторых реализациях языка SQL отобранные оператором SELECT данные могут быть сохранены в виде таблицы базы данных.

В стандарт SQL-92 включены следующие агрегатные функции: Count - подсчет, Avg - среднее, Sum - сумма, Max - максимум, Min - минимум. Запросы могут использовать **функции агрегирования**. Стандарт предусматривает использование следующих функций агрегирования:

Чаще всего функции агрегирования используются совместно с предложением GROUP BY, но *могут* применяться и самостоятельно. В последнем случае результат относится не к какой-то группе, а ко всей выборке.

В условии можно использовать агрегатные функции. Эти функции выполняются с помощью ключевых слов, которые включаются в запрос SELECT. Чтобы было понятно, вот некоторые возможности агрегатных функций в SQL:

- Суммировать выбранные значения
- Находить среднее арифметическое значений
- Находить минимальное и максимальное из значений

Наиболее часто используемые функции:

Функция SUM

Эта функция позволяет просуммировать значения какого либо поля при запросе SELECT. Достаточно полезная функция, синтаксис которой довольно прост, как и всех других агрегатных функций в SQL. Для понимания сразу начнем с примера:

Получить сумму всех заказов из таблицы Orders, которые были совершены в 2016 году.

Можно было бы просто вывести сумму заказов, но мне кажется, что это совсем просто. Напомним структуру нашей таблицы:

onum	amt	odate	cnum	snum
1001	128	2016-01-01	9	4
1002	1800	2016-04-10	10	7
1003	348	2017-04-08	2	1
1004	500	2016-06-07	3	3
1005	499	2017-12-04	5	4
1006	320	2016-03-03	5	4
1007	80	2017-09-02	7	1
1008	780	2016-03-07	1	3
1009	560	2017-10-07	3	7
1010	900	2016-01-08	6	8

Следующий код осуществит нужную выборку:

```
SELECT SUM(amt)
FROM Orders
WHERE odate BETWEEN '2016-01-01' and '2016-12-31';
```

В результате получим:

SUM(amt)

4428

В данном запросе мы использовали **функцию SUM**, после которой в скобках нужно указать поле для суммирования. Затем мы указали условие в WHERE, которое отобрало строчки только с 2016 годом. На самом деле это условие можно записать по другому, но сейчас важнее агрегатная функция суммирования в SQL.

Функция AVG

Следующая функция осуществляет подсчет среднего арифметического поля данных, которое мы укажем в качестве параметра. Синтаксис такой функции идентичен функции суммирования. Поэтому сразу перейдем к простейшей задаче:

Вывести среднюю стоимость заказа из таблицы Orders.

И сразу запрос:

```
SELECT AVG(amt)
FROM Orders;
```

В результате получим:

AVG(amt)
591.5

В целом, все похоже на предыдущую функцию. И синтаксис достаточно прост. В этом и состоит особенность языка SQL — быть понятным для человека.

Функции MIN и MAX

Еще 2 функции, которые близки по своему действию. Они находят минимальное или максимальное значение соответственно того параметра, который будет передан в скобках. Синтаксис повторяется и поэтому следующий пример:

Вывести максимальное и минимальное значения цены заказа, для тех заказов в которых цена менее 1000.

Получается такой запрос,

```
SELECT MAX(amt), MIN(amt)
FROM Orders
WHERE amt < 1000;
```

который выведет:

MAX(amt)	MIN(amt)
900	80

Также стоит сказать, что в отличие от предыдущих функций, эти 2 могут работать с символьными параметрами, то есть можно написать запрос типа **MIN(odate)** (в данном случае дата у нас символьная), и тогда нам вернется 2016-01-01.

Дело в том, что в этих функциях есть механизм преобразования символов в ASCII код, который потом они и сравнивают.

Еще одним важным моментом является то, что мы можем производить некоторые простые математические операции в запросе SELECT, например, такой запрос:

```
SELECT (MAX(amt) - MIN(amt)) AS 'Разница'
```

```
FROM Orders;
```

Вернет такой ответ:

Разница
1720

Функция COUNT

Эта функция необходима для того, чтобы подсчитать количество выбранных значений или строк. Существует два основных варианта ее использования:

- С ключевым словом DISTINCT, для того, чтобы подсчитать количество не повторяющихся значений
- С использованием «*», для того, чтобы подсчитать количество всех выбранных значений

Теперь разберем пример использования COUNT в SQL:

Подсчитать количество сделанных заказов и количество продавцов в таблице Orders.

```
SELECT COUNT(*), COUNT(DISTINCT snum)
FROM Orders;
```

Получаем:

COUNT(*)	COUNT(snum)
10	5

Очевидно, что количество заказов — 10, но если вдруг у вас имеется большая таблица, то такая функция будет очень удобной. Что касается уникальных продавцов, то здесь необходимо использовать DISTINCT, потому что один продавец может обслужить несколько заказов.

Оператор GROUP BY

Теперь рассмотрим 2 важных оператора, которые помогают расширить функционал наших запросов в SQL. Первым из них является оператор GROUP BY, который осуществляет группировку по какому либо полю, что иногда является необходимым. И уже для этой группы производит заданное действие. Например:

Вывести сумму всех заказов для каждого продавца по отдельности.

То есть теперь нам нужно для каждого продавца в таблице Orders выделить поля с ценой заказа и просуммировать. Все это сделает оператор GROUP BY в SQL достаточно легко:

```
SELECT snum, SUM(amt) AS 'Сумма всех заказов'
FROM Orders
GROUP BY snum;
```

И в итоге получим:

snum	Сумма всех заказов
1	428
3	1280
4	947
7	2360

8	900

Как видно, SQL выделил группу для каждого продавца и посчитал сумму всех их заказов.

Оператор HAVING

Этот оператор используется как дополнение к предыдущему. Он необходим для того, чтобы ставить условия для выборки данных при группировке. Если условие выполняется то выделяется группа, если нет — то ничего не произойдет. Рассмотрим следующий код:

```
SELECT snum, SUM(amt) AS 'Сумма всех заказов'
FROM Orders
GROUP BY snum
HAVING MAX(amt) > 1000;
```

Который создаст группу для продавца и посчитает сумму заказов этой группы, только в том случае, если максимальная сумма заказа больше 1000. Очевидно, что такой продавец только один, для него выделится группа и посчитается сумма всех заказов:

snum	Сумма всех заказов
7	2360

Казалось бы, почему тут не использовать условие WHERE, но SQL так построен, что в таком случае выдаст ошибку, и именно поэтому в SQL есть оператор HAVING.

Агрегатные функции используются подобно именам полей в предложении SELECT запроса, но с одним исключением, они берут имена полей как аргументы. Только числовые поля могут использоваться с SUM и AVG. С COUNT, MAX, и MIN, могут использоваться числовые или символьные пол. Когда они используются с символьными полями, MAX и MIN будут транслировать их в эквивалент ASCII, который должен сообщать, что MIN будет означать первое, а MAX последнее значение в алфавитном порядке

Примеры на агрегатные функции в SQL

1. Напишите запрос, который сосчитал бы все суммы заказов, выполненных 1 января 2016 года.

```
SELECT SUM(amt)
FROM Orders
WHERE odate = '2016-01-01';
```

2. Напишите запрос, который сосчитал бы число различных, отличных от NULL значений поля city в таблице заказчиков.

```
SELECT COUNT(DISTINCT city)
FROM customers;
```

3. Напишите запрос, который выбрал бы наименьшую сумму для каждого заказчика.

```
SELECT cnum, MIN(amt)
FROM orders
```

```
GROUP BY cnum;
```

4. Напишите запрос, который бы выбирал заказчиков, чьи имена начинаются с буквы Г.

```
SELECT cname
FROM customers
WHERE cname LIKE 'Г%' ;
```

5. Напишите запрос, который выбрал бы высший рейтинг в каждом городе.

```
SELECT city, MAX(rating)
FROM customers
GROUP BY city;
```

Список источников

– Г.Н.Федорова Основы проектирования баз данных. М.: Академия, 2020
– Г.Н.Федорова Разработка, администрирование и защита баз данных. М.: Академия, 2018
– https://infopedia.su/16x10105.html
– https://studfile.net/preview/5917121/page:2
– https://codetown.ru/category/sql