ЛЕКЦИЯ №7 ЯЗЫК SQL. ЗАПИСЬ ОПЕРАТОРОВ SQL. МАНИПУЛИРОВАНИЕ ДАННЫМИ. ПРОСТЫЕ ЗАПРОСЫ.

7.1 История SQL

Все языки манипулирования данными (ЯМД), созданные до появления реляционных баз данных и разработанные для многих систем управления базами данных (СУБД) персональных компьютеров, были ориентированы на операции с данными, представленными в виде логических записей файлов. Это требовало от пользователей детального знания организации хранения данных и достаточных усилий для указания не только того, какие данные нужны, но и того, где они размещены и как шаг за шагом получить их.

Разработка, в основном, шла в отделениях фирмы IBM (языки ISBL, SQL, QBE) и университетах США (PIQUE, QUEL). Последний создавался для СУБД INGRES (Interactive Graphics and Retrieval System), которая была разработана в начале 70-х годов в Университете шт. Калифорния и сегодня входит в пятерку лучших профессиональных СУБД. Сегодня из всех этих языков полностью сохранились и развиваются QBE (Query-By-Example - запрос по образцу) и SQL, а из остальных взяты в расширение внутренних языков СУБД только наиболее интересные конструкции.

В начале 70-х годов плодотворный труд исследователя из IBM доктора Кодда (E. F. Codd) привел к созданию продукта, связанного с реляционной моделью данных под названием SEQUEL (Structured English Query Language, структурированный английский язык для запросов), который в 1980 г. был переименован в SQL (Structured Query Language, структурированный язык запросов).

С тех пор, помимо IBM, многие производители присоединились к разработке программных продуктов для SQL. Компании IBM, а также другим производителям реляционных баз данных был нужен стандартизованный метод доступа к реляционной базе и манипулирования хранящимися в ней данными. Хотя компания IBM первая разработала теорию реляционных баз данных, первой на рынок с этой технологией вышла компания Oracle. Через какое-то время SQL завоевал на рынке достаточную популярность и привлек внимание Американского национального института по стандартизации (American National Standards Institute, ANSI), который в 1986, 1989, 1992, 1999 и 2003 годах выпустил стандарты языка SQL. Последняя версия стандарта, SQL3, содержит расширения языка для объектно-ориентированного программирования.

Начиная с 1986 года несколько конкурирующих между собой языков позволяли программистам и разработчикам обращаться к реляционным данным и манипулировать ими. Однако очень немногие из них были настолько же просты в изучении и повсеместно приняты, как SQL. Программистам и администраторам теперь можно изучить один язык, который с небольшими изменениями можно применять к разнообразным платформам баз данных, приложениям и прочим продуктам.

SQL, или язык структурированных запросов, — на сегодняшний день наиболее важный из языков манипулирования реляционными данными. Он рекомендован Американским национальным институтом стандартов (ANSI) в качестве стандартного языка манипулирования реляционными базами данных и используется как язык доступа к данным многими коммерческими СУБД, включая DB2, SQL/DS, Oracle, INGRES, SYBASE, SQL Server, dBase for Windows, Paradox, Microsoft Access и многие другие. Благодаря своей популярности SQL стал стандартным языком для обмена информацией между компьютерами. Поскольку существует версия SQL, которая может работать почти

на любом компьютере и операционной системе, компьютерные системы способны обмениваться данными, передавая друг другу запросы и ответы на языке SQL.

Постоянное развитие стандарта SQL способствовало появлению среди разных производителей и платформ многочисленных диалектов SQL. Эти диалекты развивались главным образом потому, что сообществу пользователей конкретной базы данных требовались какие-то возможности, до того как комитет ANSI разрабатывал стандарт. Однако иногда новую функциональность вводит научно-исследовательское сообщество по причине давления со стороны конкурирующих технологий. Например, многие производители баз данных к существующим возможностям программирования своих продуктов добавляют Java (как это делается в DB2, Oracle и Sybase) или VBScript (как это делает Microsoft). В будущем программисты и разработчики будут использовать эти языки программирования в SQL-программах наряду с самим SQL.

Многие из этих диалектов включают средства условной обработки (например, под контролем оператора IF...THEN), управляющие операторы (например, циклы WHILE), переменные и обработку ошибок. Поскольку ANSI пока не разработал стандарты для такой важной функциональности, а пользователи уже требуют ее, разработчики реляционных СУБД были вправе создавать свои собственные команды и синтаксис. Фактически многие из наиболее старых разработчиков сохранили с 80-х годов свои варианты самых элементарных команд (например, SELECT) поскольку их реализация предшествовала появлению стандарта. Сейчас ANSI производит уточнение стандартов, чтобы сгладить эти несоответствия.

В некоторых диалектах для обеспечения более полной функциональности языка программирования введены процедурные команды. Например, процедурные реализации содержат команды для обработки ошибок, операторы, контролирующие направление выполнения программы, условные команды, средства работы с переменными и массивами, а также многие другие дополнения. Хотя все эти процедурные реализации являются технической дивергенцией языка, здесь они называются диалектами. Пакет SQL/PSM (Persistent Stored Module) предлагает широкий спектр функциональности, связанной с хранимыми программными процедурами, и включает в себя многие расширения, содержащиеся в таких диалектах.

Вот некоторые популярные диалекты SQL:

- **PL/SQL.** Используется в Oracle. PL/SQL – это сокращение от Procedural Language/SQL. Он во многом похож на язык Ada.
- **Transact-SQL.** Используется в Microsoft SQL Server и Sybase Adaptive Server. По мере того как Microsoft и Sybase все больше отходят от общей платформы, которую они использовали в начале 90-х годов, их реализации Transact-SQL также подвергаются дивергенции.
- **PL/pgSQL.** Название диалекта и расширений SQL, реализованных в PostgreSQL. Является сокращением от Procedural Language/postgreSQL.
- **SQLPL.** Самый новый диалект от DB2 (SQLProcedural Language). Основан на стандартных операторах управления SQL. Большинство других диалектов предшествовало стандарту, и это означает, что вы найдете в них массу отличий от стандарта SQL.

7.2 Язык SQL

Язык SQL обладает следующими достоинствами:

1. независимость от конкретных СУБД. Если при создании БД не использовались нестандартные возможности языка SQL предоставляемые некоторой СУБД, то такую БД можно без изменений перенести на СУБД другого производителя. К

сожалению большинство БД используют особенности СУБД, на которой работают, что затрудняет их перенос на другую СУБД без изменений;

2. реляционная основа. Реляционная модель имеет солидный теоретический фундамент. Язык SQL основан на реляционной модели и является единственным языком для реляционных БД;
3. SQL обладает высокоуровневой структурой, напоминающей английский язык.
4. SQL позволяет создавать различные представления данных для различных пользователей;
5. SQL является полноценным языком для работы с БД;
6. стандарты языка SQL. Официальный стандарт языка SQL опубликован ANSI и ISO в 1989 году и значительно расширен в 1992 году.

Чтобы начать использовать SQL, вы должны понять, как пишутся инструкции. Синтаксические конструкции SQL делятся на четыре основные категории:

- **Идентификаторы.** Представляют собой пользовательские или системные имена объектов баз данных, таких, как база данных, таблица, ограничение в таблице, столбцы таблицы, представления и т. п.
- **Константы.** Представляют собой созданные пользователем или системой строки или значения, не являющиеся идентификаторами или ключевыми словами. Константы могут представлять собой строки, например «hello», числа, например «1234», даты, например «1 января 2002», или булевы значения, например TRUE.
- **Операторы.** Символы, показывающие, какое действие выполняется над одним или несколькими выражениями, чаще всего в инструкциях DELETE, INSERT, SELECT или UPDATE. Операторы также часто применяются для создания объектов базы данных.
- **Зарезервированные и ключевые слова.** Имеют специальный смысл для обработчика кода SQL. Например, SELECT, GRANT, DELETE или CREATE. Зарезервированные слова (Reserved words), обычно команды и инструкции SQL, нельзя использовать в качестве идентификаторов на данной платформе. Ключевые слова (keywords) - это слова, которые могут стать зарезервированными в будущем.

7.2.1 Идентификаторы

Помните, что СУРБД созданы на основе теории множеств. В терминологии ANSI кластеры содержат множество каталогов, каталоги содержат множество схем, схемы содержат множество объектов и т. д. В большинстве платформ применяются дополнительные термины: экземпляры (instances) содержат одну или несколько баз данных, базы данных содержат одну или несколько схем, схемы содержат одну или несколько таблиц, представлений, хранимых процедур и привилегий, связанных с каждым объектом. На каждом уровне структуры элементам необходимы уникальные имена (т. е., идентификаторы), чтобы к ним могли обращаться программы и системные процессы. Это означает, что каждый объект (будь то база данных, таблица, представление, столбец, индекс, ключ, триггер, хранимая процедура или ограничение) в СУРБД должен получить свой идентификатор. Если вы запускаете команду, которая создает объект базы данных, вы должны указать идентификатор (т. е. имя) этого нового объекта.

Есть два важных набора правил, которые опытный программист держит в уме при выборе имени объекта. *Соглашения об именах*

Сюда входят практические руководства или соглашения об именах, применение которых в конечном счете улучшает структуру базы и отслеживание данных. Они являются не столько требованиями SQL, сколько накопленным опытом практикующих программистов. *Правила создания идентификаторов*

Они определены в стандарте SQL и реализованы в платформах. Эти правила включают, например, такие параметры, как максимальная длина имени. Эти соглашения описываются ниже в этой главе применительно к каждому производителю.

Соглашения об именах

- *Выбирайте имя так, чтобы оно было осмысленным, наглядным и соответствовало назначению объекта.* Не называйте таблицу ХРОЗ, назовите лучше Expenses_2005, чтобы было ясно, что в ней содержатся расходы за 2005 год. Помните, что и другим людям скорее всего придется использовать таблицу и базу данных, и возможно, еще в течение долгого времени после того, как вы уйдете. Имена должны быть понятны с первого взгляда. У каждого производителя есть свои ограничения на длину имени объектов, но, как правило, имена могут быть достаточно длинными, чтобы любой мог понять их смысл.
- *Используйте в именах один и тот же регистр по всей базе*. Используйте для всех имен объектов базы либо верхний, либо нижний регистр. Некоторые серверы баз учитывают регистр, и использование смешения регистров может позже вызвать проблемы.
- *Будьте последовательны в использовании сокращений.* Как только сокращение выбрано, его нужно последовательно использовать по всей базе данных. Например, если вы используете EMP как сокращение слова EMPLOYEE, то используйте EMP по всей базе данных. Не используйте в одних местах EMP, а в других - EMPLOYEE.
- *Для удобства восприятия используйте полные, наглядные и осмысленные имена с символами подчеркивания*. Имя столбца UPPERCASEWITHLNDERSCORES не такое понятное, как UPPERCASE_WITH_UNDERSCORES.
- *Не помещайте название компании и продуктов в имена объектов баз данных.* Компании приобретают друг друга, и продукты меняют названия. Такие элементы преходящи, и их не нужно включать в имена объектов базы.
- *Не используйте слишком очевидные префиксы и суффикс*. Например, не используйте в качестве префикса имени базы данных сочетание «DB_», а в качестве префикса всех представлений - «V_». Простые запросы к системным таблицам базы могут дать администратору или программисту базы сведения о том, к какому типу относится объект, который представляет идентификатор.
- *Не заполняйте все пространство, отведенное для имени объекта.* Если платформа позволяет использовать имя таблицы из 32 символов, попробуйте оставить хотя бы несколько пустых мест в конце. Некоторые платформы при манипуляции временными копиями таблиц иногда добавляют к именам таблиц пре-фиксы и суффиксы.
- *Не используйте идентификаторы с разделителями.* Иногда имена объектов заключают в двойные кавычки. (Стандарт ANSI называет такие имена идентификаторами с разделителями (delimited identifiers). Такое заключение идентификатора в кавычки позволяет создавать имена, которые могут оказаться сложными в использовании и которые впоследствии могут вызывать проблемы. Такие идентификаторы чувствительны к регистру. Например, вы можете включать в них пробелы, специальные символы, символы в разных регистрах и даже управляющие символы. Однако, поскольку некоторые инструменты, выпускаемые сторонними производителями (и даже производителем самой базы), могут не обрабатывать специальные символы в именах, широко использовать подобные идентификаторы не следует. Некоторые платформы разрешают использовать другие символы-ограничители, помимо двойных кавычек. Например, в SQL Server

для обозначения идентификаторов с ограничителями применяются квадратные скобки [].

Правила создания идентификаторов

Идентификаторы должны быть уникальны в пределах своей области действия. Таким образом, в иерархии объектов имена баз данных не должны повторяться в пределах данного экземпляра сервера базы, а имена таблиц, представлений, функций, триггеров и хранимых процедур - уникальны в пределах данной схемы. С другой стороны, таблица и хранимая процедура могут иметь одно имя, поскольку они являются объектами разных типов. Имена столбцов, ключей и индексов должны быть уникальны в пределах одной таблицы или представления и т. д. За более подробной информацией обращайтесь к документации платформы. В некоторых платформах уникальность идентификаторов является обязательным условием, а в других - нет. Например, платформа DB2 требует, чтобы все идентификаторы индексов были уникальны по всей базе данных, а SQL Server требует, чтобы идентификаторы индексов были уникальными только в пределах таблицы, к которой они относятся.

Помните, что для обхода некоторых этих правил можно использовать идентификаторы с ограничителями (т. е. имена объектов, заключенные в специальные символы-ограничители, обычно в двойные кавычки). В частности, идентификаторы с разделителями можно применять для того, чтобы давать имена с зарезервированными словами, или для того, чтобы использовать в имени обычно не употребляемые там символы. Например, чаше всего вы не можете использовать в имени таблицы знак процента (%). Однако, если это необходимо, вы можете его использовать, если будете всегда заключать это имя таблицы в двойные кавычки. Чтобы назвать таблицу expense%%ratios, нужно заключить это имя в кавычки - "expense%%ratios". Также помните, что в SQL2003 такие имена иногда называются идентификаторами с разделителями (delimited identifiers).

7.2.2 Константы

В SQL константами считаются любые числовые значения, строки символов, значения, связанные с представлением времени (дата и время), и булевы значения, которые не являются идентификаторами или ключевыми словами. Базы данных на основе SQL разрешают использовать в коде SQL различные константы. Допустимы большинство числовых, символьных и булевых типов данных, а также даты. Например, к числовым типам данных SQL Server можно (среди прочих) отнести типы INTEGER, REAL и MONEY. Таким образом, числовые константы могут выглядеть так.

```
30
-17
-853 3888
-6.66
$70000
2E5
7E-3
```

Как показывает приведенный пример, в SQL Server допустимы числа со знаком и без знака, в обычной и экспоненциальной записи. А поскольку в SQL Server есть денежный тип данных, в константы можно включать даже знак доллара. В численных константах SQL Server не разрешается использовать другие символы (за исключением 0123456789+ -$.Ee), поэтому не включайте в них запятые (или точки, применяемые в Европе). Большинство баз данных интерпретируют запятую в числовой константе как ограничитель элементов. Так, константа 3,000 будет интерпретироваться как 3 и отдельно 000.

Булевы значения, строковые константы и даты выглядят примерно так.:

TRUE
'Hello world!'
10CT-28-1966 22:14:30:00'

Строковые константы должны всегда заключаться в одинарные кавычки ('), которые являются стандартным ограничителем всех строковых констант. Символы в строковых константах не ограничиваются алфавитными символами. По сути, любой символ из набора символов можно представить в виде строковой константы. Все приведенные ниже выражения являются строковыми константами.

'1998'
'70,000 + 14000'
'Жил-был один человек из Нантакета'
'Oct 28, 1966'

Все приведенные примеры фактически являются совместимыми с типом данных CHARACTER. Не путайте строковую константу '1998' с числовой константой 1998. Когда только строковые константы связаны с типом данных CHARACTER, не стоит использовать их в арифметических вычислениях, не преобразовав их явным образом в числовой тип. Некоторые базы данных выполняют автоматическое преобразование строковых констант, содержащих числа, при выполнении сравнения их с любыми значениями, относящимися к типам DATE или NUMBER.

При необходимости вы можете отобразить в строковой константе символ одинарной кавычки. Для этого его необходимо написать два раза; т. е., каждый раз, когда вам нужно написать внутри строки одинарную кавычку, вы должны написать две. Проиллюстрируем эту идею примером из SQL Server.

SELECT 'So he said "who"s Le Petomaine?" '

Получится следующий результат.

So he said 'Who's Le Petomaine?'

7.2.3 Операторы

Оператор - это символ, обозначающий действие, выполняемое над одним или несколькими выражениями. Операторы наиболее часто используются в инструкциях DELETE, INSERT, SELECT или UPDATE, а также часто применяются при создании объектов базы данных, таких, как хранимые процедуры, функции, триггеры и представления.

Операторы, как правило, делятся на следующие категории:

1. *Арифметические операторы*. Поддерживаются всеми базами данных.
2. *Операторы присваивания.* Поддерживаются всеми базами данных.
3. *Побитовые операторы*. Поддерживаются Microsoft SQL Server.
4. *Операторы сравнения*. Поддерживаются всеми базами данных.
5. *Логические операторы*. Поддерживаются в DB2, Oracle, SQL Server и PostgreSQL.
   Унарные операторы. Поддерживаются в DB2, Oracle и SQL Server.

Арифметические операторы

Арифметические операторы выполняют математические действия над двумя значениями любого типа, относящегося к числовой категории. Перечень арифметических операторов приведен в табл. 1.

Таблица 1 Арифметические операторы.

Арифметический оператор	**Действие**
+	Сложение
-	Вычитание
*	Умножение
/	Деление
%	Остаток отделения (только SQL Server). Возвращает остаток от операции деления в виде целого числа (integer)

Операторы присваивания

За исключением Oracle, где для этой цели применяется оператор :- оператор присваивания (=) присваивает значение переменной или псевдониму (alias) заголовка столбца. В SQL Server в качестве оператора для присваивания псевдонимов таблицам или заголовкам столбцов может служить ключевое слово AS.

Побитовые операторы

В Microsoft SQL Server существуют побитовые операторы, являющиеся удобным средством манипулирования битами в двух выражениях целого типа (см. табл. 2). Для побитовых операторов доступны следующие типы данных: *binary, hit, int, smallint, tinyint и varbinary*.

Таблица 2 Побитовые операторы

Побитовый оператор	**Действие**
&	Поразрядное И (два операнда)
|	Поразрядное ИЛИ (два операнда)
^	Поразрядное исключающее ИЛИ (два операнда)

Операторы сравнения

Операторы сравнения проверяют равенство или неравенство двух выражений. Результатом операции сравнения является булево значение: TRUE, FALSE или UNKNOWN. Также заметьте, что по стандарту ANSI сравнение выражений, когда одно или оба значения равны NULL, дает результат NULL. Например, выражение 23 + NULL дает NULL, как и выражение Feb 23, 2003 + NULL.

Таблица 3 Операторы сравнения

Оператор сравнения	**Действие**
=	Равно
>	Больше
<	Меньше
>=	Больше или равно
<=	Меньше или равно

<>	Не равно
! =	Не равно (не соответствует стандарту ANSI)
!<	Не меньше (не соответствует стандарту ANSI)
!>	Не больше (не соответствует стандарту ANSI)

Логические операторы

Логические операторы обычно применяются в предложении WHERE для проверки истинности какого-либо условия. Логические операторы возвращают булево значение TRUE или FALSE. Логические операторы также обсуждаются в разделе «Инструкция SELECT» главы 3. Не все базы данных поддерживают все операторы. Список логических операторов приведен в табл. 4.

Таблица 4 Логические операторы

Логический оператор	Действие
ALL	TRUE, если весь набор сравнений дает результат TRUE
AND	TRUE, если оба булевых выражения дают результат TRUE
ANY	TRUE, если хотя бы одно сравнение из набора дает результат TRUE
BETWEEN	TRUE, если операнд находится внутри диапазона
EXISTS	TRUE, если подзапрос возвращает хотя бы одну строку
IN	TRUE, если операнд равен одному выражению из списка или одной или нескольким строкам, возвращаемым подзапросом
LIKE	TRUE, если операнд совпадает с шаблоном
NOT	Обращает значение любого другого булевого оператора
OR	TRUE, если любое булево выражение равно TRUE
SOME	TRUE, если несколько сравнений из набора дают результат TRUE

Унарные операторы

Унарные операторы выполняют операцию над одним выражением любого типа, относящимся к числовой категории. Унарные операторы можно применять к целым типам, хотя операторы положительности и отрицательности можно применять к любому числовому типу данных (см. табл. 5).

Таблица 5 Унарные операторы

Унарный оператор	Действие
+	Числовое значение становится положительным
-	Числовое значение становится отрицательным
~	Поразрядное НЕ, возвращает двоичное дополнение числа (пет в Oracle и DB2)

Приоритет операторов

Иногда выражения, включающие операторы, могут быть довольно сложными. Когда в выражении присутствуют несколько операторов, последовательность их выполне-

ния определяет приоритет операторов. Порядок выполнения может существенно повлиять на результат вычисления.

Ниже перечислены уровни приоритета операторов. Оператор с более высоким приоритетом выполняется до оператора с более низким приоритетом. В списке операторы перечислены в порядке от самого высокого к самому низкому приоритету. () (выражения, стоящие в скобках)

1. +, -, ~ (унарные операторы)
2. *, /, % (математические операторы)
3. +, - (арифметические операторы)
4. =, >, <, >=, <=, <>, !=, !>, !< (операторы сравнения)
5. ^ (побитовое исключающее ИЛИ), & (побитовое И), | (побитовое ИЛИ)
6. NOT, AND, ALL, ANY, BETWEEN IN LIKE, OR, SOME = (присваивание значение переменной)

Если операторы имеют одинаковый приоритет, вычисления производятся слева направо. Для того чтобы изменить применяемый по умолчанию порядок выполнения операторов, в выражении используются скобки. Выражения, заключенные в скобки, вычисляются первыми, а уже после них - все, что находится за скобками.

Строковые ограничители обозначают границы строки, состоящей из буквенно-цифровых символов. ***Системные ограничители*** - это такие символы из набора символов, которые имеют для сервера базы данных особое значение. ***Ограничители*** - это символы, которые применяются для определения иерархического порядка процессов и элементов списка. ***Операторы*** - это ограничители, используемые для определения значений при операциях сравнения, в том числе символы, обычно используемые для арифметических и математических операций. В табл. 6 перечислены системные ограничители и операторы, допустимые в SQL.

7.2.4 Ключевые и зарезервированные слова

Наряду с символами, которые имеют особый смысл и функции в SQL, существуют и некоторые слова и фразы, имеющие особую значимость. ***Ключевые слова SQL*** - это слова, которые настолько тесно связаны с функционированием реляционной базы данных, что их нельзя использовать ни для каких других целей. Как правило, такие слова используются в инструкциях SQL. (Заметьте, что на большинстве платформ их можно использовать в качестве идентификаторов, хотя этого делать не следует.) Например, слово «SELECT» является зарезервированным словом, и его не следует использовать в качестве имени таблицы.

С другой стороны, зарезервированные слова в настоящий момент не имеют специального назначения, но они могут приобрести его в будущем. Чтобы подчеркнуть тот факт, что ключевые слова не следует использовать в качестве идентификаторов, но тем не менее такая возможность существует, в стандарте SQL они называются «незарезервированными ключевыми словами». Зарезервированные и ключевые слова SQL не всегда представляют собой слова, используемые в инструкциях SQL, они также могут быть связаны с технологией использования базы данных. Например, слово CASCADE применяется для описания таких манипуляций с данными, в которых действие (например, удаление или обновление) распространяется «вниз», т. е. «каскадом» на все нижележащие таблицы. Зарезервированные и ключевые слова часто публикуются, чтобы программисты не использовали их в качестве идентификаторов и чтобы в дальнейшем, в следующих версиях, не возникали проблемы.

7.2.5 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.

Чаще всего выделяются два языка - язык определения схемы БД (SDL - Schema Definition Language) и язык манипулирования данными (DML - Data Manipulation Language). SDL служил главным образом для определения логической структуры БД, т.е. той структуры БД, какой она представляется пользователям. DML содержал набор операторов манипулирования данными, т.е. операторов, позволяющих заносить данные в БД, удалять, модифицировать или выбирать существующие данные.

7.3 Язык определения данных

В состав языка определения входят операторы:

1. CREATE – создает объектов базы данных
2. ALTER – изменяет объект
3. DROP –удаляет объект

Стандарт SQL-92 определяет команды для следующих объектов:

1. ASSERTION – утверждения для проверки
2. CHARACTER SET – набор символов
3. COLLATION – правила сортировки для набора символов
4. DOMAIN – домен (пользовательского типа данных столбца).
5. SCHEMA – схема (именованной группы объектов)
6. TABLE – таблица базы данных
7. TRANSLATION – правила преобразования (трансляции) из одного набора символов в другой (используется в операторе TRANSLATE)
8. VIEW – представления данных

7.3.1 Типы данных

ANSI SQL включает в себя следующие типы данных

Символьные строки:

- `CHARACTER(`*n*`)`или `CHAR(`*n*`)` — строка фиксированной длины в *n* символов, разделенная пробелами;
- `CHARACTER VARYING(`*n*`)`или `VARCHAR(`*n*`)` — строка переменной длины с максимальным количеством символов *n*;
- `NATIONAL CHARACTER(`*n*`)`или `NCHAR(`*n*`)` — строка фиксированной длины с поддержкой международных кодировок
- `NATIONAL CHARACTER VARYING(`*n*`)`или `NVARCHAR(`*n*`)` — строка переменной длины `NCHAR`.

Битовые данные:

- `BIT(`*n*`)` — массив из *n* битов
- `BIT VARYING(`*n*`)` — массив длинной до *n* битов

Числа:

- `INTEGER` и `SMALLINT` — целые числа;
- `FLOAT`, `REAL` и `DOUBLE PRECISION` — вещественные числа;
- `NUMERIC(`*precision*`, `*scale*`)`или `DECIMAL(`*precision*`, `*scale*`)`— вещественное число с указанием в скобках количество знаков до запятой и после запятой.

Дата и время:

- `DATE` — дата (`2010-05-30`);
- `TIME` — время (`14:55:37`);
- `TIME WITH TIME ZONE` или `TIMESTAMP` — тоже самое, что и `TIME`, только исключаются данные о часовом поясе;
- `TIMESTAMP` — это `DATE` и `TIME` соединенные вместе в одной переменной (`2010-05-30 14:55:37`).
- `TIMESTAMP WITH TIME ZONE` or `TIMESTAMPTZ` — тоже самое, что и `TIMESTAMP`, только исключаются данные о часовом поясе.

7.3.2 Домен

Создание домена:

```
CREATE DOMAIN <имя_домена> [AS] <тип_данных>
[DEFAULT {LITERAL | NULL | USER}]
  [NOT NULL] [CHECK (<условие>)]
  [COLLATE <имя_сортировки>];
```

где

- ***DEFAULT*** – Определяет значение по умолчанию, которое вставляется, когда ни какой другой ввод не сделан. Значения:
  - ***LITERAL*** – Вводится определенная строка, числовое значение или дата.
  - ***NULL*** – Вводится значение NULL.
  - ***USER*** – Вводится имя текущего пользователя. Столбец должен быть совместимый символьный тип, что бы использовать значение по умолчанию.
- ***NOT NULL*** – Определяет, что значения введенные в столбец не могут быть NULL.
- ***CHECK*** – (<условие>) создает одиночное CHECK ограничение для домена.
- ***VALUE*** – Заменитель для имени столбца, в конечном счете, основанном на домене.
- ***COLLATE <имя_сортировки>*** – Устанавливает способ сортировки для домена.

Следующая инструкция создает домен, который может принимать положительные значения больше 1000, со значением по умолчанию 9999. Ключевое слово VALUE заменяется именем столбца основанном на этом домене.

```
CREATE DOMAIN CUSTNO
  AS INTEGER
    DEFAULT 9999
    CHECK (VALUE > 1000);
```

Следующая инструкция ограничивает введенные значения в домен до четырех определенных значений:

```
CREATE DOMAIN PRODTYPE
  AS VARCHAR(12)
    CHECK (VALUE IN ("software", "hardware", "other", "N/A"));
```

Изменение домена:

```
ALTER DOMAIN <имя_домена> {
[SET DEFAULT {LITERAL | NULL | USER}]
  | [DROP DEFAULT]
  | [ADD [CONSTRAINT] CHECK (<условие>)]
  | [DROP CONSTRAINT]
  };
```

где

- ***SET DEFAULT*** – Определяет значение столбца по умолчанию, которое будет введено, когда ни какой другой ввод не сделан. Значения:
  - ***LITERAL*** – Вводится определенная строка, числовое значение или дата.

- ***NULL –*** Вводится значение NULL.
- ***USER –*** Вводится имя текущего пользователя. Столбец должен быть совместимого текстового типа для использования значения по умолчанию.

Установка значения по умолчанию на уровне столбца отменяет установку значения по умолчанию на уровне домена.

- ***DROP DEFAULT*** – Удаляет существующее значение по умолчанию.
- ***ADD [CONSTRAINT] CHECK (<условие>)*** – Добавляет CHECK ограничения в определение домена. Определение домена может включать только одно CHECK ограничение.
- ***DROP CONSTRAINT*** – Удаляет CHECK ограничения из определения домена.

Следующая инструкция устанавливает значение домена по умолчанию к 9999.

```
ALTER DOMAIN CUSTNO SET DEFAULT 9999;
```

Удаление домена:

```
DROP DOMAIN <имя_домена>;
```

7.3.2 Схема

Создание базы данных:

```
CREATE {DATABASE | SCHEMA} <имя_базы_данных>
```

Удаление базы данных:

```
DROP {DATABASE | SCHEMA} <имя_базы_данных>
```

7.3.4 Таблица

Создание таблицы

```
CREATE TABLE [ IF NOT EXISTS ] <имя_таблицы>
(
<имя столбца_1> <тип данных> [ DEFAULT expression ] [ {NULL | NOT NULL} ] [
{INDEX_BLIST | INDEX_NONE} ]

<имя столбца_2> <тип данных> [ DEFAULT expression ] [ {NULL | NOT NULL} ] [
{INDEX_BLIST | INDEX_NONE} ]

...

<имя столбца_N> <тип данных> [ DEFAULT expression ] [ {NULL | NOT NULL} ] [
{INDEX_BLIST | INDEX_NONE} ]

[ CONSTRAINT <имя ограничения> ]
PRIMARY KEY ( <имя столбца_1>, <имя столбца_2>, ... ) |

FOREIGN KEY (<имя столбца_1>, <имя столбца_2>, ... ) REFERENCES
<имя_таблицы_2> [ (<имя столбца_1>, <имя столбца_2>, ... ) ] [ ON UPDATE {NO
ACTION | SET NULL | SET DEFAULT | CASCADE} ] [ ON DELETE {NO ACTION | SET
NULL | SET DEFAULT | CASCADE} ] |

UNIQUE (<имя столбца_1>, <имя столбца_2>, ... ) |

CHECK ( <условие> ) [ {INITIALLY DEFERRED | INITIALLY IMMEDIATE} ] [ {NOT
DEFERRABLE | DEFERRABLE} ]
);
```

где

- ***DEFAULT*** expression – значение по умолчанию;
- ***NULL*** | ***NOT NULL*** – разрешается ли пустое поле;
- ***INDEX_BLIST*** | ***INDEX_NONE*** – есть или нет индекса;

- **CONSTRAINT** – ограничение
  - ***PRIMARY KEY*** **–** первичный ключ
  - ***FOREIGN KEY*** **–** вторичный ключ
    - ***ON DELETE*** **–** при удалении в родительской таблице
    - ***ON UPDATE*** **–** при обновлении в родительской таблице
    - ***NO ACTION*** **–** нет действий
    - ***SET NULL*** **–** устанавливается значение NULL
    - ***SET DEFAULT*** **–** устанавливается значение по умолчанию
    - ***CASCADE*** **–** каскадно
  - ***UNIQUE*** **–** уникальный
  - ***CHECK*** **–** проверка

Например

```
CREATE TABLE Customer (
  number VARCHAR(40)  NOT NULL,
  name   VARCHAR(100) NOT NULL,
  ssn    VARCHAR(50)  NOT NULL,
  age    INTEGER      NOT NULL,

  CONSTRAINT cust_pk PRIMARY KEY (number),
  UNIQUE ( ssn ),              // (An anonymous constraint)
  CONSTRAINT age_check CHECK (age >= 0 AND age < 200)
);
```

Изменение таблицы:

Переименование таблицы

```
ALTER TABLE <имя_таблицы> RENAME TO <новое_имя_таблицы>
```

Переименование столбца

```
ALTER TABLE <имя_таблицы> RENAME [ COLUMN ] <имя_столбца> TO
<новое_имя_столбца>
```

Добавление столбца

```
ALTER TABLE <имя_таблицы> ADD [COLUMN] <имя_столбца> <тип_данных> [ DEFAULT
expression ] [ {NULL | NOT NULL} ] [ {INDEX_BLIST | INDEX_NONE} ]
```

Добавление первичного ключа ограничения к таблице

```
ALTER TABLE <имя_таблицы> ADD [ CONSTRAINT <имя ограничения> ]
PRIMARY KEY ( <имя столбца_1>, <имя столбца_2>, ... ) |
```

Добавление вторичного ключа ограничения к таблице

```
ALTER TABLE <имя_таблицы> ADD [ CONSTRAINT <имя ограничения> ]
FOREIGN KEY (<имя столбца_1>, <имя столбца_2>, ... ) REFERENCES
<имя_таблицы_2> [ (<имя столбца_1>, <имя столбца_2>, ... ) ] [ ON UPDATE {NO
ACTION | SET NULL | SET DEFAULT | CASCADE} ] [ ON DELETE {NO ACTION | SET
NULL | SET DEFAULT | CASCADE} ] |
```

Добавление уникального поля к таблице

```
ALTER TABLE <имя_таблицы> ADD [ CONSTRAINT <имя ограничения> ]
UNIQUE (<имя столбца_1>, <имя столбца_2>, ... ) |
```

Добавление проверки столбца к таблице

```
ALTER TABLE <имя_таблицы> ADD [ CONSTRAINT <имя ограничения> ]
CHECK ( <условие> ) [ {INITIALLY DEFERRED | INITIALLY IMMEDIATE} ] [ {NOT
DEFERRABLE | DEFERRABLE} ]
```

Изменение типа данных столбца

```
ALTER TABLE <имя_таблицы> MODIFY "column 1" "New Data Type"
```

Изменение столбца ограничений столбца

```
ALTER TABLE <имя_таблицы> ALTER [COLUMN] column_name SET default_expr
ALTER TABLE <имя_таблицы> ALTER [COLUMN] column_name DROP DEFAULT
```

Изменение удаление столбца

```
ALTER TABLE <имя_таблицы> DROP [COLUMN] column_name
```

Удаление ограничения таблицы

```
ALTER TABLE <имя_таблицы> DROP CONSTRAINT constraint_name
```

Удаление первичного ключа

```
ALTER TABLE <имя_таблицы> DROP PRIMARY KEY
```

Удаление таблицы

```
DROP TABLE [ IF EXISTS ] <имя_таблицы>
```

7.4 Язык манипулирования данными.

В основу языка манипулирования данными входят 4 основных оператора:

1. SELECT – используется для выборки записей из таблиц;
2. INSERT –используется для добавления записей в таблицу;
3. UPDATE – используется для обновления записей таблицы;
4. DELETE – используется для удаления записей из таблицы.

В самой простой форме, команда SELECT просто инструктирует базу данных чтобы извлечь информацию из таблицы.

7.4.1 Оператор INSERT

Все строки в SQL вводятся с использованием команды модификации INSERT. В самой простой форме, INSERT использует следующий синтаксис:

```
INSERT INTO <имя_таблицы> [(<имя столбца_1> [, <имя столбца_1> ...])]
   {VALUES (<значение_1> [, <значение_2> ...]) | <выражение SELECT>};
```

Так, например, чтобы ввести строку в таблицу Продавцов, вы можете использовать следующее условие:

```
INSERT INTO Salespeople VALUES (1001, 'Peel', 'London', .12);
```

Команды DML не производят никакого вывода, но ваша программа должна дать вам некоторое подтверждение того что данные были использованы.

Вы можете также указывать столбцы, куда вы хотите вставить значение имени. Это позволяет вам вставлять имена в любом порядке. Предположим что вы берете значения для таблицы Заказчиков из отчета выводимого на принтер, который помещает их в таком порядке: city, cname, и cnum, и для упрощения, вы хотите ввести значения в том же порядке:

```
INSERT INTO Customers (city, cname, cnum) VALUES ('London', 'Honman', 2001);
```

Обратите внимание что столбцы rating и snum - отсутствуют. Это значит, что эти строки автоматически установлены в значение - по умолчанию. По умолчанию может быть введено или значение NULL или другое значе- ние определяемое как - по умолчанию. Если ограничение запрещает использование значения NULL в данном столбце, и этот столбец не установлен как по умолчанию, этот столбец должен быть обеспечен значением для любой команды INSERT которая относится к таблице.

7.4.2 Оператор UPDATE

Теперь, вы должны узнать как изменять некоторые или все значения в существующей строке. Это выполняется командой UPDATE. Эта команда содержит предложение UPDATE в которой указано им используемой таблицы и предложение SET которое указывает на изменение которое нужно сделать для определенного столбца. Например, чтобы изменить оценки всех заказчиков на 200, вы можете ввести

```
UPDATE TABLE <имя_таблицы>
   SET <имя столбца_1> = <значение_1> [, <имя столбца_2> = <значение_2> ...]
   [WHERE <условие>];
```

Например

```
UPDATE Customers
   SET rating = 200;
```

Конечно, вы не всегда захотите указывать все строки таблицы для изменения единственного значения, так что UPDATE может брать предикаты. Вот как например можно выполнить изменение одинаковое для всех заказчиков продавца Peel (имеющего snum=1001):

```
UPDATE Customers
   SET rating = 200
   WHERE snum = 1001;
```

Однако, вы не должны, ограничивать себя модифицированием единственного столбца с помощью команды UPDATE. Предложение SET может назначать любое число столбцов, отделяемых запятыми. Все указанные назначения могут быть сделаны для любой табличной строки, но только для одной в каждый момент времени. Предположим, что продавец Motika ушел на пенсию, и мы хотим переназначить его номер новому продавцу:

```
UPDATE Salespeople
   SET sname = 'Gibson',city = 'Boston',comm = .10
   WHERE snum = 1004;
```

Вы можете использовать скалярные выражения в предложении SET команды UPDATE, однако, включив его в выражение пол которое будет изменено. В этом их отличие от предложения VALUES команды INSERT, в котором выражения не могут использоваться; это свойство скалярных выражений - весьма полезна особенность. Предположим, что вы решили удвоить комиссионные всем вашим продавцам. Вы можете использовать следующее выражение:

```
UPDATE Salespeople
   SET comm = comm * 2;
```

7.4.3 Оператор DELETE

Вы можете удалять строки из таблицы командой модификации - DELETE. Она может удалять только введенные строки, а не индивидуальные значения полей.

```
DELETE FROM table [WHERE <search_condition>];
```

Теперь когда таблица пуста ее можно окончательно удалить командой DROP TABLE (это объясняется в Главе 17). Обычно, вам нужно удалить только некоторые определенные строки из таб- лицы. Чтобы определить какие строки будут удалены, вы используете предикат, так же как вы это делали для запросов. Например, чтобы удалить продавца под номером 1003 из таблицы, вы можете ввести

```
DELETE FROM Salespeople
   WHERE snum = 1003;
```

Мы использовали поле snum вместо пол sname потому, что это лучшая тактика при использовании первичных ключей когда вы хотите чтобы действию подвергалась одна и только одна строка. Для вас - это аналогично действию первичного ключ. Конечно, вы можете также использовать DELETE с предикатом который бы выбирал группу строк, как показано в этом примере:

```
DELETE FROM Salespeople
   WHERE city = 'London';
```

7.4.4 Оператор SELECT

Все запросы в SQL состоят из одиночной команды. Структура этой команды обманчиво проста, потому что вы должны расширять ее так чтобы выполнить высоко сложные оценки и обработки данных. Эта команда называется - SELECT(ВЫБОР).

```
SELECT  [[ALL] | DISTINCT]{ * | элемент_SELECT [,элемент_SELECT] ...}
FROM            {базовая_таблица | представление} [псевдоним]
                [,{базовая_таблица | представление} [псевдоним]] ...
[WHERE  фраза]
[GROUP BY фраза [HAVING фраза]];
[ORDER BY фраза]
```

Если вы хотите видеть каждый столбец таблицы, имеется необязательное сокращение которое вы можете использовать. Звездочка (*) может применяться для вывода полного списка столбцов следующим образом:

```
SELECT * FROM Salespeople;
```