Лекция 3. Общая характеристика оператора SELECT и организация списка ссылок на таблицы в разделе FROM

- Введение
- Скалярные выражения
  - Общие синтаксические правила построения скалярных выражений
  - Численные выражения
  - Выражения, значениями которых являются символьные или битовые строки
  - Выражения даты-времени
  - Булевские выражения
  - Выражения с переключателем
- Общая структура оператора выборки в языке SQL
  - Семантика оператора выборки
  - Ссылки на таблицы раздела FROM
  - Табличное выражение, спецификация запроса и выражение запросов
  - Раздел WITH выражения запросов
  - Конструкторы значения строки и таблицы
  - Ссылки на базовые, представляемые и порождаемые таблицы
  - Представляемые таблицы, или представления (VIEW)

Введение

- В этой и следующих трех лекциях рассматривается важнейший оператор языка SQL – оператор SELECT, предназначенный для выборки данных из SQL-ориентированной базы данных
- Этот оператор имеет довольно сложную и развитую структуру
- его необходимо знать любому специалисту, так или иначе связанному с использованием баз данных; поэтому ему уделяется так много внимания
- Первая лекция носит подготовительный характер
  - виды скалярных выражений, используемые, прежде всего, в конструкциях оператора SELECT
  - базовая семантика выполнения этого оператора
  - принципы и разновидности указания таблиц, из которых производится выборка данных

Скалярные выражения (1)

- *Скалярное выражение* – это выражение, вырабатывающее результат некоторого типа, специфицированного в стандарте
- Скалярные выражения являются основой языка SQL, поскольку все условия, элементы списков выборки и т. д. базируются именно на скалярных выражениях
- В SQL имеется несколько разновидностей скалярных выражений
- К числу наиболее важных разновидностей относятся
  - численные выражения;
  - выражения со значениями-строками символов;
  - выражения со значениями даты-времени;
  - выражения со значениями-временными интервалами;
  - булевские выражения
- Приведем некоторые базовые спецификации и пояснения.

Скалярные выражения (2)

- **Общие синтаксические правила построения скалярных выражений**
- В SQL:2003 имеются девять разновидностей выражений в соответствии с девятью категориями типов данных, значения которых вырабатываются при вычислении выражения

```
value_expression ::=
    numeric_value_expression
  | string_value_expression
  | datetime_value_expression
  | interval_value_expression
  | boolean_value_expression
  | array_value_expression
  | multiset_value_expression
  | row_value_expression
  | user_defined_value_expression
  | reference_value_expression
```

- Ограничимся обсуждением первых пяти разновидностей выражений

Скалярные выражения (3)

- В основе построения этих видов выражений лежит первичное выражение, определяемое следующим синтаксическим правилом:

```
value_expression_primary ::=
    unsigned_value_specification
  | column_reference
  | set_function_specification
  | scalar_subquery
  | case_expression
  | (value_expression)
  | cast_specification
```

- В пределах этого курса можно считать, что спецификация беззнакового значения (unsigned_value_specification) – это всегда литерал соответствующего типа или вызов *ниладической* функции (например, CURRENT_USER)

Скалярные выражения (4)

```
value_expression_primary ::=
    unsigned_value_specification
  | column_reference
  | set_function_specification
  | scalar_subquery
  | case_expression
  | (value_expression)
  | cast_specification
```

- При вычислении выражения V для строки таблицы каждая ссылка на столбец (column_reference) этой таблицы, непосредственно содержащаяся в V, рассматривается как ссылка на значение данного столбца в данной строке
- Агрегатные функции (*функции над множествами* – set_function_specification) обсуждаются позже
- Если первичное выражение является скалярным подзапросом
  - scalar_subquery - подзапросом, результатом которого является таблица, состоящая из одной строки и одного столбца

  и результат подзапроса пуст,
  - то результат первичного выражения – неопределенное значение

Скалярные выражения (5)

- **Численные выражения**
- *Численное выражение* – это выражение, значение которого относится к числовому типу данных. Вот формальный синтаксис численного выражения:

```
numeric_value_expression> ::= numeric_term
    | numeric_value_expression + term
    | numeric_value_expression - term
numeric_term ::= numeric_factor
    | numeric_term * numeric_factor
    | numeric_term / numeric_factor
numeric_factor ::= [ { + | - } ] numeric_primary
numeric_primary ::= value_expression_primary
    | numeric_value_function
```

- В численных выражениях SQL первичная составляющая (numeric_primary) является либо первичным выражением, либо вызовом функции с численным значением (numeric_value_function)
  - из этого следует, что в численные выражения могут входить выражения с переключателем и операции преобразования типов

Скалярные выражения (6)

- Вызовы функций с численным значением определяются следующими синтаксическими правилами:

```
numeric_value_function ::=
   POSITION (character_value_expression
   IN character_value_expression)
   |{CHAR_LENGTH | CHARACTER_LENGTH }
   (string_value_expression)
   | OCTET_LENGTH (string_value_expression)
   | BIT_LENGTH (string_value_expression)
   | EXTRACT ({ datetime_field | time_zone field }
     FROM { datetime_value_expression
       | interval_value_expression })
   | CARDINALITY (array_value_expression
         | multiset_value_expression)
   | ABS (numeric_value_expression)
   | MOD (numeric_value_expression)
```

- Достаточно подробно обсуждали функции определения позиции и длины по отношению к символьным и битовым строкам при рассмотрении соответствующих типов данных
  - здесь приводится только уточненный синтаксис их вызова

Скалярные выражения (7)

```
numeric_value_function ::=
   POSITION (character_value_expression
   IN character_value_expression)
   |{CHAR_LENGTH | CHARACTER_LENGTH }
   (string_value_expression)
   | OCTET_LENGTH (string_value_expression)
   | BIT_LENGTH (string_value_expression)
   | EXTRACT ({ datetime_field | time_zone field }
     FROM { datetime_value_expression
       | interval_value_expression })
   | CARDINALITY (array_value_expression
         | multiset_value_expression)
   | ABS (numeric_value_expression)
   | MOD (numeric_value_expression)
```

- Функция EXTRACT извлечения поля из значений дата-время или интервал позволяет получить в виде точного числа с масштабом 0 значение любого поля (года, месяца, дня и т. д.)
  - какой конкретный тип точных чисел будет выбран – определяется в реализации
- Функции ABS и MOD возвращают абсолютное значение числа и остаток от деления одного целого значения на другое соответственно

Скалярные выражения (8)

- **Выражения, значениями которых являются символьные или битовые строки**
- Выражения символьных и битовых строк – это выражения, значениями которых являются символьные или битовые строки
- Соответствующие конструкции определяются следующим синтаксисом:

  string_value_expression ::=

  character_value_expression | bit_value_expression
  character_value_expression ::= concatenation | character_factor
  concatenation ::=
  character_value_expression || character_factor

  character_factor ::= character_primary [collate_clause]
  character_primary ::=

  value_expression_primary | string_value_function

  bit_value_expression ::= bit_concatenation | bit_factor
  bit_concatenation ::= bit_value_expression || bit_primary bit_primary ::= value_expression_primary | string value function

Скалярные выражения (9)

- Если не вдаваться в тонкости, смысл выражений символьных и битовых строк понятен из описания синтаксиса:
  - единственная применимая для построения выражений операция – это конкатенация, производящая «склейку» строк-операндов
- Более важно то, что первичной составляющей выражения над строками может быть как первичное скалярное выражение, так и вызов функций, возвращающих строчные значения

Скалярные выражения (10)

- Репертуар и синтаксис вызова таких функций определяются следующими правилами:

```
string_value_function ::= character_value_function
   | bit_value_function
character _value_function ::= SUBSTRING
   (character _value_expression
     FROM start_position
       [ FOR string_length ])
     | SUBSTRING (character _value_expression
       SIMILAR character _value_expression
         ESCAPE character_value_expression)
     | { UPPER | LOWER }
       (character_value_expression)
     | CONVERT (character_value_expression
       USING conversion_name)
     | TRANSLATE (character_value_expression)
       USING translation_name)
     | TRIM ([ {LEADING | TRAILING | BOTH} ]
       [ character_value_expression ]
       [ character_value_expression ])
     | OVERLAY (character_value_expression
       PLACING character_value_expression
         FROM start_position
           [ FOR string_length ])
bit _value_function ::= SUBSTRING (bit_value_expression
       FROM start_position
         [ FOR string_length ])
start_position ::= numeric_value_expression
string length ::= numeric_value_expression
```

Скалярные выражения (11)

- Основные полезные функции – выделение подстроки (SUBSTRING) и замена малых букв на заглавные и наоборот (UPPER и LOWER) – мы упоминали при рассмотрении типов символьных и битовых строк
- Обсуждение функции SUBSTRING ... SIMILAR ... ESCAPE отложим до следующей лекции
- Для символьных строк имеются еще четыре функции: CONVERT, TRANSLATE, TRIM и OVERLAY
- Функция CONVERT меняет кодировку символов в заданной строке,
  - причем набор символов не меняется
- Способ задания правил перекодировки определяется в реализации. Функция TRANSLATE в соответствии с правилами трансляции «переводит» текстовую строку на другой язык
  - используя набор символов целевого алфавита
  - кодировка не меняется
- Функция TRIM «отсекает» последовательности указанного символа в начале, в конце или в конце и начале заданной строки
- Функция OVERLAY заменяет указанную подстроку первого операнда строкой, заданной в качестве второго операнда

Скалярные выражения (12)

- **Выражения даты-времени**
- К выражениям даты-времени мы относим выражения, вырабатывающие значения типа дата-время и интервал
- Выражения даты-времени определяются следующими синтаксическими правилами:

```
datetime_value_expression ::=
   datetime_term
     | interval_value_expression + datetime term
     | datetime_value_expression + interval term
     | datetime value expression - interval term
datetime_term ::= datetime_primary
   [ AT { LOCAL | TIME ZONE interval_value_expression } ]
datetime_primary ::= value_expression_primary
     | datetime_value_function
```

- Выражения строятся очень просто – на основе обычных арифметических операций
- Более интересны первичные составляющие – вызовы функций, возвращающих значение дата-время

Скалярные выражения (13)

- Эти вызовы определяются следующим синтаксисом:

```
datetime_value_function ::= CURRENT_DATE
   | CURRENT_TIME [ (precision) ]
   | LOCALTIME [ (precision) ]
   | CURRENT_TIMESTAMP [ (precision) ]
   | LOCALTIMESTAMP [ (precision) ]
```

- Приведенные синтаксические правила не нуждаются в комментариях:
  - можно получить текущую дату, а также текущее время с желаемой точностью
- Отличие функций LOCALTIME и LOCALTIMESTAMP от CURRENT_TIME и CURRENT_TIMESTAMP, соответственно, состоит в том, что
  - первая пара функций не возвращает смещение локального времени от Гринвича

Скалярные выражения (14)

- Синтаксис выражений со значениями типа интервал определяется следующими правилами:

```
interval_value_expression ::=
     interval_term
   | interval_value_expression + interval term
   | interval_value_expression - interval term
   | (datetime value expression - datetime term)
     interval_qualifier
interval_term ::= interval_factor
   | interval_term * numeric_factor
   | interval_term / numeric_factor
   | numeric_term * interval_factor

interval_factor ::= [ { + | - } ]
     interval_primary [ <interval qualifier> ]
interval_primary ::= value_expression_primary
   | interval_value_function
```

- Квалификатор интервала указывается для того, чтобы явно специфицировать единицу измерения интервала
- Поддерживается только одна функция ABS (абсолютное значение), аргументом которой является выражение со значением типа интервал

Скалярные выражения (15)

- **Булевские выражения**
- К булевским выражениям относятся выражения, вырабатывающие значения булевского типа
- булевский тип языка SQL содержит три логических значения – true, false и unknown
- Булевские выражения определяются следующими синтаксическими правилами:

```
boolean_value_expression ::= boolean_term
   | boolean_value_expression OR boolean_term
boolean_term ::= boolean_factor
   | boolean_term AND boolean_factor
boolean_factor ::= [ NOT ] boolean_test
boolean_test ::= boolean_primary [ IS [ NOT ] truth_value ]
truth_value ::= TRUE | FALSE | UNKNOWN
boolean_primary ::= predicate
   | (boolean_value_expression)
   | value_expression_primary
```

Скалярные выражения (16)

- Выражения вычисляются слева направо с учетом приоритетов операций и круглых скобок
- Операции IS и IS NOT определяются следующими таблицами истинности:

IS	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	FALSE
FALSE	FALSE	TRUE	FALSE
UNKNOWN	FALSE	FALSE	TRUE

IS NOT	TRUE	FALSE	UNKNOWN
TRUE	FALSE	TRUE	TRUE
FALSE	TRUE	FALSE	TRUE
UNKNOWN	TRUE	TRUE	FALSE

Скалярные выражения (17)

- Разные выражения с переключателем могут вырабатывать значения разных типов в зависимости от типа данных элементов
- Начнем с синтаксиса:

  case_expression ::= case_abbreviation | case_specification

  case_abbreviation ::= NULLIF (value_expression, value_expression) | COALESCE (value_expression_comma_list)

  case specification ::= simple_case | searched_case

  simple_case ::= CASE value_expression simple_when_clause_list [ELSE value_expression] END

  searched_case ::= CASE searched_when_clause_list [ELSE value_expression] END

  simple_when_clause ::= WHEN value_expression THEN value_expression

  searched_when_clause ::= WHEN conditional_expression THEN value_expression

Скалярные выражения (18)

- Наиболее общим видом выражения с переключателем является выражение с поисковым переключателем (searched_case)
  searched_case ::= CASE searched_when_clause_list
  [ELSE value_expression] END

  searched_when_clause ::= WHEN conditional_expression THEN value_expression
- Правила вычисления выражений этого вида состоят в следующем
- Вычисляется логическое выражение, указанное в первом разделе WHEN списка (searched_when_clause_list)
- Если значение этого логического выражения равняется *true*, то
  - значением всего выражения с поисковым переключателем является значение выражения, указанного в первом разделе WHEN после ключевого слова THEN
- Иначе аналогичные действия производятся для второго раздела WHEN и т. д.

Скалярные выражения (19)

- **searched_case** ::= CASE searched_when_clause_list
  [ELSE value_expression] END

  searched_when_clause ::= WHEN conditional_expression THEN
  value_expression
- Если ни для одного раздела WHEN при вычислении логического выражения не было получено значение *true*, то
  - значением всего выражения с поисковым переключателем является значение выражения, указанного в разделе ELSE
- Типы всех выражений, значения которых могут являться результатом выражения с поисковым переключателем, должны быть совместимыми, и
  - типом результата является «наименьший общий» тип набора типов выражений-кандидатов на выработку результата
- Если в выражении отсутствует раздел ELSE, предполагается наличие раздела ELSE NULL

Скалярные выражения (20)

simple_case ::= CASE value_expression simple_when_clause_list [ELSE value_expression] END
simple_when_clause ::= WHEN value_expression THEN value_expression

- В выражении с простым переключателем (simple_case) тип данных операнда переключателя
  - выражения, непосредственно следующего за ключевым словом CASE, назовем его CO – Case Operand

должен быть совместим с типом данных операнда каждого варианта

  - выражения, непосредственно следующего за ключевым словом WHEN; назовем WO – When Operand

Скалярные выражения (21)

- Выражение с простым переключателем

```
CASE CO WHEN WO1 THEN result1
   WHEN WO2 THEN result2
   . . . . . .
   WHEN WOn THEN resultn
   ELSE result
END
```

- эквивалентно выражению с поисковым переключателем

```
CASE WHEN CO = WO1 THEN result1
   WHEN CO = WO2 THEN result2
   . . . . . .
   WHEN CO = WOn THEN resultn
   ELSE result
END
```

Скалярные выражения (22)

- Выражение NULLIF (V1, V2) эквивалентно следующему выражению с переключателем:

  CASE WHEN V1 = V2 THEN NULL ELSE V1 END
- Выражение COALESCE (V1, V2) эквивалентно следующему выражению с переключателем:

  CASE WHEN V1 IS NOT NULL THEN V1 ELSE V2 END
- Выражение COALESCE (V1, V2, . . . V*n*) для $n \geq 3$ эквивалентно следующему выражению с переключателем:
  CASE WHEN V1 IS NOT NULL THEN V1
  ELSE COALESCE (V2,... n) END

Общая структура оператора выборки в языке SQL (1)

- Для выборки данных в прямом SQL используется оператор SELECT, возвращающий
  - набор из одной или нескольких строк одинаковой структуры

и задаваемый в следующем синтаксисе:

```
SELECT [ ALL | DISTINCT ] select_item_commalist
   FROM table_reference_commalist
     [ WHERE conditional_expression ]
       [ GROUP BY column_name_commalist ]
         [ HAVING conditional_expression ]
           [ ORDER BY order_item_commalist ]
```

Общая структура оператора выборки в языке SQL (2)

- **Семантика оператора выборки**
- Для начала опишем общую схему выполнения оператора SELECT в соответствии с предписаниями стандарта
- Выполнение запроса состоит из нескольких шагов, соответствующих разделам оператора выборки

```
SELECT [ ALL | DISTINCT ] select_item_commalist
   FROM table_reference_commalist
     [ WHERE conditional_expression ]
       [ GROUP BY column_name_commalist ]
         [ HAVING conditional_expression ]
           [ ORDER BY order_item_commalist ]
```

- На первом шаге выполняется раздел FROM
  - Если список ссылок на таблицы (table_reference_commalist) этого раздела соответствует таблицам A, B, … C, то в результате выполнения раздела FROM образуется таблица
    - назовем ее T,
  - являющаяся расширенным декартовым произведением таблиц A, B, …, C
  - Если в разделе FROM указана только одна таблица, то она же и является результатом выполнения этого раздела

Общая структура оператора выборки в языке SQL (3)

- В реляционной алгебре для корректного выполнения операции взятия расширенного декартова произведения отношений требуется применение операции переименования атрибутов
  - Соответствующие возможности переименования столбцов таблиц, указанных в списке раздела FROM, поддерживаются и в SQL

```
SELECT [ ALL | DISTINCT ] select_item_commalist
   FROM table_reference_commalist
     [ WHERE conditional_expression ]
       [ GROUP BY column_name_commalist ]
         [ HAVING conditional_expression ]
           [ ORDER BY order_item_commalist ]
```

- Альтернативный способ именования столбцов результирующей таблицы T основывается на использовании квалифицированных имен столбцов
  - Идея этого подхода (более раннего в истории SQL) заключается в том, что с любой таблицей, ссылка на которую содержится в списке раздела FROM, можно связать некоторое имя-псевдоним
    - в стандарте оно называется correlation name
  - Тогда если с такой таблицей A связан псевдоним Z, то в пределах оператора выборки можно ссылаться на любой столбец a таблицы A по квалифицированному имени Z.a
- Пока же будем считать, что имена всех столбцов таблицы T определены и различны

Общая структура оператора выборки в языке SQL (4)

```
SELECT [ ALL | DISTINCT ] select_item_commalist
   FROM table_reference_commalist
     [ WHERE conditional_expression ]
       [ GROUP BY column_name_commalist ]
         [ HAVING conditional_expression ]
           [ ORDER BY order_item_commalist ]
```

- На втором шаге выполняется раздел WHERE
  - Условное выражение (conditional_expression) этого раздела применяется к каждой строке таблицы T, и результатом является таблица T1, содержащая те и только те строки таблицы T, для которых результатом вычисления условного выражения является true
    - Заголовки таблиц T и T1 совпадают
  - Если раздел WHERE в операторе выборки отсутствует, то это трактуется как наличие раздела WHERE true,
    - т. е. T1 содержит те и только те строки, которые содержатся в таблице T
  - Обратите внимание на разницу в трактовке логических выражений в операторах выборки и в табличных ограничениях целостности
  - Логическое выражение раздела WHERE (и раздела HAVING) оператора выборки разрешает выборку строки в том и только в том случае, когда результатом вычисления логического выражения на данной строке является true
    - значения false и uknown не являются разрешающими
  - Логическое выражение табличного ограничения целостности запрещает наличие строки в таблице в том и только в том случае, когда результатом вычисления логического выражения на данной строке является false
    - значения true и uknown не являются запрещающими

Общая структура оператора выборки в языке SQL (5)

```
SELECT [ ALL | DISTINCT ] select_item_commalist
   FROM table_reference_commalist
     [ WHERE conditional_expression ]
       [ GROUP BY column_name_commalist ]
         [ HAVING conditional_expression ]
           [ ORDER BY order_item_commalist ]
```

- Если в операторе выборки присутствует раздел GROUP BY, то он выполняется на третьем шаге
  - Каждый элемент списка имен столбцов (column_name_commalist), указываемого в этом разделе, должен быть одним из имен столбцов таблицы T1
  - В результате выполнения раздела GROUP BY образуется *сгруппированная таблица* T2, в которой строки таблицы T1 расставлены в минимальное число групп, таких, что во всех строках одной группы значения столбцов, указанных в списке имен столбцов раздела GROUP BY
    - *столбцов группировки*,

    одинаковы
  - Сгруппированные таблицы не могут являться окончательным результатом оператора выборки
    - они существуют только на концептуальном уровне на стадии выполнения запроса, содержащего раздел GROUP BY

Общая структура оператора выборки в языке SQL (6)

- При наличии в запросе раздела HAVING, которому не предшествует раздел GROUP BY, таблица T1 рассматривается как
  - сгруппированная таблица, состоящая из одной группы строк, без столбцов группирования

```
SELECT [ ALL | DISTINCT ] select_item_commalist
   FROM table_reference_commalist
     [ WHERE conditional_expression ]
       [ GROUP BY column_name_commalist ]
         [ HAVING conditional_expression ]
           [ ORDER BY order_item_commalist ]
```

- В этом случае логическое выражение раздела HAVING может состоять только из предикатов с агрегатными функциями, а результат вычисления этого раздела T3 либо совпадает с таблицей T1, либо является пустым.
- Если в операторе выборки присутствует раздел GROUP BY, но отсутствует раздел HAVING, то это трактуется как
  - наличие раздела HAVING true, т. е. T3 содержит те и только те группы строк, которые содержатся в таблице T2

Общая структура оператора выборки в языке SQL (7)

- Рассмотрим, каким образом формируются значения столбцов в таблице T4
- Элемент списка выборки может задаваться одним из двух способов:

```
select_item ::= value_expression [ [ AS ] column_name ]
   | [ correlation_name . ] *
```

- Сначала обсудим первый вариант
- В этом случае каждый элемент списка элементов выборки соответствует столбцу таблицы T4
- Столбцу может быть явным образом приписано имя
- Порядок формирования значения этого столбца для выделенных выше случаев (a) и (b) различается, и мы рассмотрим подобные случаи по отдельности

Общая структура оператора выборки в языке SQL (8)

```
select_item ::= value_expression [ [ AS ] column_name ]
   | [ correlation_name . ] *
```

- Если сгруппированная таблица T3 была образована за счет наличия раздела HAVING без присутствия раздела GROUP BY, то в выражении элемента выборки
  - вообще нельзя непосредственно использовать имена столбцов таблицы T3
- Имена других столбцов таблицы T3 могут использоваться только в конструкциях вызова агрегатных функций COUNT, SUM, MIN, MAX, AVG
- Выражение вычисляется для каждой группы строк таблицы T3
- Именам столбцов, входящих в выражение непосредственно, сопоставляются значения этих столбцов, которые соответствуют данной группе зстрок таблицы T3

Общая структура оператора выборки в языке SQL (9)

```
select_item ::= value_expression [ [ AS ] column_name ]
   | [ correlation_name . ] *
```

- Во втором варианте спецификация элемента списка выборки вида [Z.]* является
  - сокращенной формой записи списка Z.a1, Z.a2, …, Z.a*n*, где
    - a1, a2, …, a*n* представляет собой полный список имен столбцов таблицы, псевдоним которой Z
- Следует сделать три замечания
  - Во-первых, для именованной таблицы, входящей в список раздела FROM только один раз, можно использовать имя таблицы вместо псевдонима
  - Во-вторых, во втором варианте спецификации элемента списка выборки можно опустить псевдоним только в том случае, если в разделе FROM указана только одна таблица
  - В-третьих, в случае (b) второй вариант спецификации элемента выборки допустим только тогда, когда все столбцы таблицы с псевдонимом Z входят в список столбцов группировки раздела GROUP BY

Общая структура оператора выборки в языке SQL (10)

```
SELECT [ ALL | DISTINCT ] select_item_commalist
   FROM table_reference_commalist
     [ WHERE conditional_expression ]
       [ GROUP BY column_name_commalist ]
         [ HAVING conditional_expression ]
           [ ORDER BY order_item_commalist ]
```

- Итак, мы получили таблицу T4
- Если в спецификации раздела SELECT
  - отсутствует ключевое слово DISTINCT, или
  - присутствует ключевое слово ALL,
  - либо отсутствуют и ALL, и DISTINCT,

  то T4 является результатом выполнения раздела SELECT
- В противном случае на завершающей стадии выполнения раздела SELECT в таблице T4 удаляются строки-дубликаты

Общая структура оператора выборки в языке SQL (11)

- Если в операторе выборки не содержится раздел ORDER BY, то таблица T4 является
  - результирующей таблицей запроса
- Иначе на завершающей стадии выполнения запроса производится сортировка строк таблицы T4
  - в соответствии со списком элементов сортировки (order_item_commalist) раздела ORDER BY
- В стандарте SQL:1999 элемент списка элементов сортировки имеет следующую синтаксическую форму:

```
SELECT [ ALL | DISTINCT ] select_item_commalist
   FROM table_reference_commalist
     [ WHERE conditional_expression ]
       [ GROUP BY column_name_commalist ]
         [ HAVING conditional_expression ]
           [ ORDER BY order_item_commalist ]
```

```
order_item ::= value_expression [ collate_clause ]
   [ { ASC | DESC } ]
```

Общая структура оператора выборки в языке SQL (12)

- Выполнение раздела ORDER BY производится следующим образом
  - Выбирается первый элемент списка сортировки, и строки таблицы T4 расставляются
    - в порядке возрастания
      - если в элементе присутствует спецификация ASC;
        - при отсутствии спецификации ASC/DESC предполагается наличие ASC
    - или в порядке убывания
      - при наличии спецификации DESC

    в соответствии со значениями выражения, содержащегося в данном элементе, которые вычисляются для каждой строки таблицы T4
  - Далее выбирается второй элемент списка сортировки, и в соответствии со значениями заданного в нем выражения и порядка сортировки расставляются строки, которые после первого шага сортировки образовали
    - группы с одинаковым значением выражения первого элемента списка сортировки
  - Операция продолжается до исчерпания списка элементов сортировки
  - Результирующий отсортированный список строк является окончательным результатом запроса

```
order_item ::= value_expression [ collate_clause ]
   [ { ASC | DESC } ]
```

Общая структура оператора выборки в языке SQL (13)

- В общем случае выражение, входящее в элемент списка сортировки, основывается

```
order_item ::= value_expression [ collate_clause ]
   [ { ASC | DESC } ]
```

- на именах столбцов таблицы T4 и именах столбцов таблицы, над которой вычислялся раздел SELECT (T1 или T3)
- Идея состоит в том, что
- если некоторое выражение могло бы быть использовано в элементе списка выборки, то
- его можно использовать в элементе списка сортировки
- В стандарте SQL:1999 присутствует ряд чисто технических ограничений на вид выражений, допустимых в элементах списка сортировки,
- если в запросе присутствуют разделы GROUP BY и/или HAVING
- и если в разделе SELECT присутствует спецификация DISTINCT
- Но в любом случае это выражение может иметь вид a, где a – имя столбца таблицы T4

Общая структура оператора выборки в языке SQL (13)

- В общем случае выражение, входящее в элемент списка сортировки, основывается

```
order_item ::= value_expression [ collate_clause ]
   [ { ASC | DESC } ]
```

  - на именах столбцов таблицы T4 и именах столбцов таблицы, над которой вычислялся раздел SELECT (T1 или T3)
- Идея состоит в том, что
  - если некоторое выражение могло бы быть использовано в элементе списка выборки, то
  - его можно использовать в элементе списка сортировки
- В стандарте SQL:1999 присутствует ряд чисто технических ограничений на вид выражений, допустимых в элементах списка сортировки,
  - если в запросе присутствуют разделы GROUP BY и/или HAVING
  - и если в разделе SELECT присутствует спецификация DISTINCT
- Но в любом случае это выражение может иметь вид a, где a – имя столбца таблицы T4

Общая структура оператора выборки в языке SQL (14)

- В предыдущих версиях стандарта языка SQL, включая SQL/92, элемент списка сортировки определялся следующим синтаксическим правилом:

```
order_item ::= { column_name | unsigned_integer }
   [ { ASC | DESC } ]
```

- В качестве имени столбца (column_name) можно было использовать любое имя, вводимое для столбца таблицы T4 в элементе списка выборки
- Вместо имени столбца можно было использовать его порядковый номер (unsigned_integer) в списке элементов выборки раздела SELECT
- В новом стандарте вторая возможность исключена
  - Доводом является не тот факт, что использование номеров столбцов противоречит реляционной модели
  - Использование номеров столбцов запрещено, поскольку не давало возможности применять в элементах списка сортировки выражения
- Тем не менее возможность использования номеров столбцов в течение долгого времени будет продолжать поддерживаться в коммерческих реализациях SQL,
  - поскольку она применяется во многих существующих приложениях

Общая структура оператора выборки в языке SQL (15)

- **Ссылки на таблицы раздела FROM**
- Рассмотрим более подробно, какой вид могут иметь элементы списка ссылок на таблицы в разделе FROM
- Для начала приведем полный набор синтаксических правил SQL:1999, определяющий table_reference

table_reference ::= table_primary | joined_table
table_primary ::=
table_or_query_name [[AS] correlation_name [(derived_column_list)]] |
derived_table [[AS] correlation_name [(derived_column_list)]] |
lateral_derived_table [[AS] correlation_name [(derived_column_list)]] |
collection_derived_table [[AS] correlation_name [(derived_column_list)]] | ONLY (table_or_query_name) [[AS] correlation_name [(derived_column_list)]] | (joined_table)
table_or_query_name ::= { table_name | query_name }
derived_table ::= (query_expression)
lateral_derived_table ::= LATERAL (query_expression)
collection_derived_table ::= UNNEST (collection_value_exression) [WITH ORDINALITY]

Общая структура оператора выборки в языке SQL (15)

- Отложим обсуждение порождаемых таблиц с горизонтальной связью (lateral_derived_table) и «соединенных таблиц» (joined_table)
- Кроме того, мы не будем рассматривать в этом курсе конструкции collection_derived_table и ONLY (table_or_query_name),
- поскольку они относятся к объектным расширениям языка SQL, которые в данном курсе подробно не рассматриваются
- Но все равно дальнейшего продвижения нам придется определить несколько дополнительных синтаксических конструкций языка SQL

Общая структура оператора выборки в языке SQL (16)

- **Табличное выражение, спецификация запроса и выражение запросов**
- *Табличным выражением* (table_expression) называется конструкция

```
table_expression ::= FROM table_reference_commalist
   [ WHERE conditional_expression ]
     [ GROUP BY column_name_commalist ]
       [ HAVING conditional_expression ]
```

- *Спецификацией запроса* (query_specification) называется конструкция

```
query_specification SELECT [ ALL | DISTINCT ]
   select_item_commalist table_expression
```

Общая структура оператора выборки в языке SQL (17)

- Наконец, *выражением запросов* (query_expression) называется конструкция

```
query_expression ::= [ with_clause ] query_expression_body
query_expression_body ::= { non_join_query_expression
   | joined_table }
non_join_query_expression ::= non_join_query_term
   | query_expression_body
     { UNION | EXCEPT }[ ALL | DISTINCT ]
       [ corresponding_spec ] query_term
query_term ::= non_join_query_term | joined_table
non_join_query_term ::= non_join_query_primary
   | query_term INTERSECT [ ALL | DISTINCT ]
     [ corresponding_spec ] query_primary
query_primary ::= non_join_query_primary | joined_table
non_join_query_primary ::= simple_table
   | (non_join_query_expression)
simple_table ::= query_specification
   | table_value_constructor
   | TABLE table_name
corresponding_spec ::= CORRESPONDING
   [ BY column_name_comma_list ]
```

Общая структура оператора выборки в языке SQL (18)

- Если не обращать внимания на не обсуждавшиеся пока конструкции joined_table и table_value_constructor, синтаксические правила показывают, что

```
query_expression_body ::= { non_join_query_expression
   | joined_table }
non_join_query_expression ::= non_join_query_term
   | query_expression_body
     { UNION | EXCEPT }[ ALL | DISTINCT ]
       [ corresponding_spec ] query_term
```

  - выражение запросов строится из выражений, значениями которых являются таблицы, с использованием «теоретико-множественных» операций UNION (объединение), EXCEPT (вычитание) и INTERSECT (пересечение)
- Операция пересечения является «мультипликативной» и обладает более высоким приоритетом, чем «аддитивные» операции объединения и вычитания
- Вычисление выражения производится слева направо с учетом приоритетов операций и круглых скобок

Общая структура оператора выборки в языке SQL (19)

- При этом действуют следующие правила

```
query_expression_body ::= { non_join_query_expression
   | joined_table }
non_join_query_expression ::= non_join_query_term
   | query_expression_body
     { UNION | EXCEPT }[ ALL | DISTINCT ]
       [ corresponding_spec ] query_term
```

- Если выражение запросов не включает ни одной теоретико-множественной операции,
  - то результатом вычисления выражения запросов является результат вычисления простой или соединенной таблицы
- Если в терме (non_join_query_term) или выражении запросов (non_join_query_expression) без соединения присутствует теоретико-множественная операция, то
  - пусть T1, T2 и TR обозначают соответственно первый операнд, второй операнд и результат терма или выражения соответственно,
  - а OP – используемую теоретико-множественную операцию

Общая структура оператора выборки в языке SQL (20)

- Если в операции присутствует спецификация CORRESPONDING, то:

```
query_expression_body ::= { non_join_query_expression
   | joined_table }
non_join_query_expression ::= non_join_query_term
   | query_expression_body
     { UNION | EXCEPT }[ ALL | DISTINCT ]
       [ corresponding_spec ] query_term
```

  - если присутствует конструкция BY column_name_comma_list, то все имена в этом списке должны быть различны, и каждое имя должно являться одновременно именем некоторого столбца таблицы T1 и именем некоторого столбца таблицы T2, причем типы этих столбцов должны быть совместимыми; обозначим данный список имен через SL;
  - если список соответствия столбцов не задан, пусть SL обозначает список имен столбцов, являющихся именами столбцов и в T1, и в T2, в том порядке, в котором эти имена фигурируют в T1;
  - вычисляемые терм или выражение запросов без соединения эквивалентны выражению (SELECT SL FROM T1) OP (SELECT SL FROM T2), не включающему спецификацию CORRESPONDING.

Общая структура оператора выборки в языке SQL (21)

- При отсутствии в операции спецификации CORRESPONDING операция выполняется таким образом,

```
query_expression_body ::= { non_join_query_expression
   | joined_table }
non_join_query_expression ::= non_join_query_term
   | query_expression_body
     { UNION | EXCEPT }[ ALL | DISTINCT ]
       [ corresponding_spec ] query_term
```

  - как если бы эта спецификация присутствовала и включала конструкцию BY column_name_comma_list, в которой были бы перечислены все столбцы таблицы T1
- При выполнении операции OP две строки s1 с именами столбцов c1, c2, …, c*n* и s2 с именами столбцов d1, d2, …, d*n* считаются строками-дубликатами,
  - если для каждого *i* (*i* = 1, 2, …, *n*)
    - либо c*i* и d*i* не содержат NULL, и (c*i* = d*i*) = *true*,
    - либо и c*i*, и d*i* содержат NULL

Общая структура оператора выборки в языке SQL (22)

- Если в операции OP не задана спецификация ALL, то

```
query_expression_body ::= { non_join_query_expression
   | joined_table }
non_join_query_expression ::= non_join_query_term
   | query_expression_body
     { UNION | EXCEPT }[ ALL | DISTINCT ]
       [ corresponding_spec ] query_term
```

  - в TR строки-дубликаты удаляются
- Если спецификация ALL задана, то
  - пусть s – строка, являющаяся дубликатом некоторой строки T1, или некоторой строки T2, или обеих;
  - пусть *m* – число дубликатов s в T1, а *n* – число дубликатов s в T2
  - тогда:
    - если указана операция UNION, то число дубликатов s в TR равно *m* + *n*;
    - если указана операция EXCEPT, то число дубликатов s в TR равно max ((*m*-*n*),0);
    - если указана операция INTERSECT, то число дубликатов s в TR равно min (*m*,*n*)

Общая структура оператора выборки в языке SQL (23)

- **Раздел WITH выражения запросов**

```
query_expression ::= [ with_clause ] query_expression_body
```

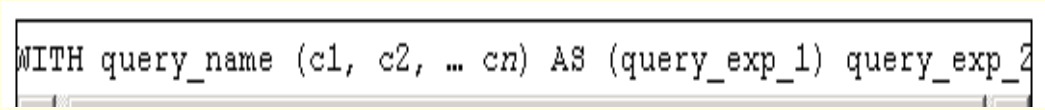
- Как видно из синтаксиса выражения запросов, в этом выражении может присутствовать раздел WITH. Он задается в следующем синтаксисе:

```
with_clause ::= WITH [ RECURSIVE ] with_element_comma_list
with_element ::= query_name [ (column_name_list) ]
   AS (query_expression) [ search_or_cycle_clause ]
```

- Ограничимся случаем, когда в разделе WITH отсутствуют спецификация RECURSIVE и search_or_cycle_clause

Общая структура оператора выборки в языке SQL (24)

- Тогда конструкция



означает, что в любом месте выражения запросов query_exp_2, где допускается появление ссылки на таблицу, можно использовать имя query_name

- Можно считать, что перед выполнением query_exp_2 происходит выполнение query_exp_1, и результирующая таблица с именами столбцов c1, c2, … cn сохраняется под именем query_name
- В этом случае раздел WITH фактически служит для локального определения представляемой таблицы (VIEW)

Общая структура оператора выборки в языке SQL (25)

- **Конструкторы значения строки и таблицы**
- Чтобы завершить обсуждение выражений запросов
  - конструкция соединенных таблиц (joined_table) отложена

  нам осталось рассмотреть конструкции table_value_constructor и TABLE table_name
- В определении конструктора значения-таблицы используется конструктор значения-строки, который строит упорядоченный набор скалярных значений, представляющий строку
  - возможно и использование подзапроса

row_value_constructor ::= row_value_constructor_element |
[ROW] (row_value_constructor_element_comma_list) |
row_subquery
row_value_constructor_element ::= value_expression |
NULL |
DEFAULT

Общая структура оператора выборки в языке SQL (26)

- Значение элемента по умолчанию можно использовать только в том случае, когда конструктор значения-строки применяется в операторе INSERT
  - тогда этим значением будет значение по умолчанию соответствующего столбца
- Конструктор значения-таблицы производит таблицу на основе заданного набора конструкторов значений-строк:

```
table_value_constructor ::= VALUES
   row_value_constructor_comma_list
```

- Для корректного построения таблицы требуется,
  - чтобы строки, производимые всеми конструкторами строк, были одной и той же степени и
  - чтобы типы (или домены) соответствующих столбцов являлись приводимыми
- Конструкция TABLE table_name является сокращенной формой записи выражения SELECT * FROM table_name

Общая структура оператора выборки в языке SQL (27)

- **Ссылки на базовые, представляемые и порождаемые таблицы**
- Теперь мы можем завершить обсуждение разновидностей ссылок на таблицу в разделе FROM
- Для удобства повторим синтаксические правила
  - опустив конструкции, рассмотрение которых отложено
  - или выходит за пределы материала данного курса:

table_reference ::= table_primary
table_primary ::= table_or_query_name [[AS] correlation_name [(derived_column_list)]] | derived_table [AS] correlation_name [(derived_column_list)]
table_or_query_name ::= { table_name | query_name } derived_table ::= (query_expression)

- В самом простом случае в качестве ссылки на таблицу используется *имя таблицы*
  - *базовой* или *представляемой*
- или имя запроса, присоединенного к данному запросу с помощью раздела WITH
- В другом случае
  - derived_table

порождаемая таблица задается выражением запроса, заключенным в круглые скобки

Общая структура оператора выборки в языке SQL (28)

table_reference ::= table_primary

table_primary ::= table_or_query_name [[AS] correlation_name
[(derived_column_list)]] | derived_table [AS] correlation_name
[(derived_column_list)]

table_or_query_name ::= { table_name | query_name }

derived_table ::= (query_expression)

- Явное указание имен столбцов результата запроса из раздела WITH или порождаемой таблицы требуется в том случае, когда
  - эти имена не выводятся явно из соответствующего выражения запроса
- В таких случаях в соответствующем элементе списка раздела FROM должен указываться псевдоним (correlation_name),
  - потому что иначе таблица была бы вообще лишена имени
- Можно считать, что выражение запроса вычисляется и сохраняется во временной таблице при обработке раздела FROM

Общая структура оператора выборки в языке SQL (29)

- Может смутить рекурсивная природа синтаксических определений, приведенных в этом подразделе
- Чтобы определить понятие ссылки на таблицу в разделе FROM оператора выборки, который опирается на спецификацию запроса, нам пришлось ввести более общее понятие выражения запросов, в определении которого используется спецификация запроса
- Да, действительно, многие синтаксические конструкции SQL определяются рекурсивно
- Но эта рекурсия никогда не приводит к зацикливанию
- В частности, раскрутка рекурсии операторов выборки основывается на базовой, не выделяемой отдельными синтаксическими правилами форме, в которой в разделе FROM указываются только имена базовых таблиц

Общая структура оператора выборки в языке SQL (30)

- **Представляемые таблицы, или представления (VIEW)**
- Еще одним примером рекурсивности спецификаций языка SQL является то, что в конце этой лекции мы вынуждены прервать обсуждение оператора выборки и ввести понятие
  - представляемой таблицы, или представления,

  которую можно использовать в операторе выборки наряду с базовыми таблицами
- Только после этого можно будет считать обсуждение ссылок на таблицы в разделе FROM условно завершенным
- Итак, оператор создания представления в общем случае определяется следующими синтаксическими правилами:

```
create_view ::= CREATE [ RECURSIVE ] VIEW table_name
     [ column_name_comma_list ]
   AS query_expression
   [ WITH [ CASCADED | LOCAL ] CHECK OPTION ]
```

Общая структура оператора выборки в языке SQL (31)

- Рекурсивные представления
  - такие, в определении которых присутствует ключевое слово RECURSIVE
- и необязательный раздел WITH CHECK OPTION отложим
  - этот раздел связан с особенностями выполнения операций обновления базы данных через представления
- Рассмотрим только простую форму представлений, определяемых по следующим правилам:

```
create_view ::= CREATE VIEW table_name
     [ column_name_comma_list ]
   AS query_expression
```

Общая структура оператора выборки в языке SQL (32)

- Имя таблицы, задаваемое в определении представления, существует в том же пространстве имен, что и имена базовых таблиц, и, следовательно, должно отличаться от всех имен таблиц
  - базовых и представляемых,

  созданных тем же пользователем
- Если имя представления встречается в разделе FROM какого-либо оператора выборки, то
  - вычисляется выражение запроса, указанное в разделе AS, и
  - оператор выборки работает с результирующей таблицей этого выражения запроса
- Явное указание имен столбцов представляемой таблицы требуется в том случае, когда эти имена не выводятся из соответствующего выражения запроса

Общая структура оператора выборки в языке SQL (33)

- Как и для всех других вариантов оператора CREATE, для CREATE VIEW имеется обратный оператор DROP VIEW table_name, выполнение которого приводит к отмене определения представления
  - реально это выражается в удалении данных о представлении из таблиц-каталогов базы данных
- После выполнения операции пользоваться представлением с данным именем становится невозможно
- Конструкция ALTER VIEW в языке SQL не поддерживается