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)

- 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

```
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 (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

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

  - 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

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
  - на именах столбцов таблицы T4 и именах столбцов таблицы, над которой вычислялся раздел SELECT (T1 или T3)

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

- 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
```

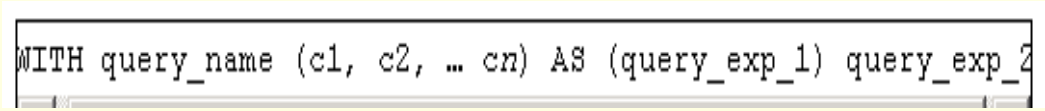
- Как видно из синтаксиса выражения запросов, в этом выражении может присутствовать раздел WITH. Он задается в следующем синтаксисе:

```
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 (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

Общая структура оператора выборки в языке SQL (34)

- **Предикаты раздела WHERE оператора SELECT**
- **Логические выражения раздела WHERE**
- Синтаксически логическое выражение раздела WHERE определяется как булевское выражение (boolean_value_expression)
- Основой логического выражения являются предикаты
- Предикат позволяет специфицировать условие, результатом вычисления которого может быть true, false или unknown

- predicate ::= comparison_predicate
  | between_predicate
- | null_predicate
  | in_predicate
  | like_predicate
- | similar_predicate
- | exists_predicate
  | unique_predicate
  | overlaps_predicate
- | quantified_comparison_predicate
  | match_predicate
  | distinct_predicate

EMP:

EMP_NO : EM_NO
EMP_NAME : VARCHAR
EMP_BDATE : DATE
EMP_SAL : SALARY
DEPT_NO : DEPT_NO
PRO_NO : PRO_NO

DEPT:

DEPT_NO : DEPT_NO
DEPT_NAME : VARCHAR
DEPT_EMP_NO : INTEGER
DEPT_TOTAL_SAL : SALARY
DEPT_MNG : EMP_NO

PRO:

PRO_NO : PRO_NO
PRO_TITLE : VARCHAR
PRO_SDATE : DATEP
PRO_DURAT : INTERVAL
PRO_MNG : EMP_NO
PRO_DESC : CLOB

Общая структура оператора выборки в языке SQL (35)

- **Предикат сравнения**

```
comparison_predicate ::=
    row_value_constructor comp_op row_value_constructor
comp_op ::= = | <> («не равно») | < | >
    | <= («меньше или равно») | >= («больше или равно»)
```

- Строки, являющиеся операндами операции сравнения, должны быть одинаковой степени
- Типы данных соответствующих значений строк-операндов должны быть совместимы
- Пусть X и Y обозначают соответствующие элементы строк-операндов, а xv и yv – их значения. Тогда:
  - если xv и/или yv являются неопределенными значениями, то значение условия X comp_op Y - unknown;
  - в противном случае значением условия X comp_op Y является true или false в соответствии с естественными правилами применения операции сравнения.

Общая структура оператора выборки в языке SQL (36)

- Числа сравниваются в соответствии с правилами алгебры
- Сравнение двух символьных строк производится следующим образом:
  - если длина строки X не равна длине строки Y, то для выравнивания длин строк более короткая строка расширяется *символами набивки* (*pad symbol*); если для используемого набора символов порядок сортировки явным образом не специфицирован, то в качестве символа набивки используется пробел;
  - далее производится лексикографическое сравнение строк в соответствии с предопределенным или явно определенным порядком сортировки символов
- Сравнение двух битовых строк X и Y основано на сравнении соответствующих бит. Если X_i и Y_i – значения *i*-тых бит X и Y соответственно и если lx и ly обозначает длину в битах X и Y соответственно, то:
  - X равно Y тогда и только тогда, когда lx = ly и $X_i = Y_i$ для всех *i*;
  - X меньше Y тогда и только тогда, когда (a) lx < ly и $X_i = Y_i$ для всех *i* меньших или равных lx, или (b) $X_i = Y_i$ для всех $i < n$ и $X_n = 0$, а $Y_n = 1$ для некоторого *n* меньшего или равного min (lx, ly)
- Сравнение двух значений типа дата-время производится в соответствии с видом интервала, который получается при вычитании второго значения из первого
  - Пусть X и Y – сравниваемые значения, а H – наименее значимое поле даты-времени X и Y
  - Результат сравнения X comp_op Y определяется как (X – Y) H comp_ op INTERVAL (0) H
  - Два значения типа дата-время сравнимы только в том случае, если они содержат одинаковый набор полей даты-времени

Общая структура оператора выборки в языке SQL (37)

- Сравнение двух значений анонимного строкового типа производится следующим образом
- Пусть Rx и Ry обозначают строки-операнды, а Rx_i и Ry_i – i-тые элементы Rx и Ry соответственно
- Вот как определяется результат сравнения Rx comp_op Ry:
  - Rx = Ry есть true тогда и только тогда, когда $Rx_i = Ry_i$ есть true для всех *i*;
  - Rx <> Ry есть true тогда и только тогда, когда $Rx_i <> Ry_i$ есть true для некоторого *i*;
  - Rx < Ry есть true тогда и только тогда, когда $Rx_i = Ry_i$ есть true для всех $i < n$, и $Rx_n < Ry_n$ есть true для некоторого *n*;
  - Rx > Ry есть true тогда и только тогда, когда $Rx_i = Ry_i$ есть true для всех $i < n$, и $Rx_n > Ry_n$ есть true для некоторого *n*;
  - Rx <= Ry есть true тогда и только тогда, когда Rx = Ry есть true или Rx < Ry есть true;
  - Rx >= Ry есть true тогда и только тогда, когда Rx = Ry есть true или Rx > Ry есть true;
  - Rx = Ry есть false тогда и только тогда, когда Rx <> Ry есть true;
  - Rx <> Ry есть false тогда и только тогда, когда Rx = Ry есть true;
  - Rx < Ry есть false тогда и только тогда, когда Rx >= Ry есть true;
  - Rx > Ry есть false тогда и только тогда, когда Rx <= Ry есть true;
  - Rx <= Ry есть false тогда и только тогда, когда Rx > Ry есть true;
  - Rx >= Ry есть false тогда и только тогда, когда Rx < Ry есть true;
  - Rx comp_op Ry есть unknown тогда и только тогда, когда Rx comp_op Ry не есть true или false

Общая структура оператора выборки в языке SQL (38)

- **Примеры запросов с использованием предиката сравнения**
- Найти номера отделов, в которых работают служащие с фамилией 'Smith‘

```
SELECT DISTINCT EMP.DEPT_NO
FROM EMP
WHERE EMP.EMP_NAME = 'Smith';
```

- DISTINCT в разделе SELECT, потому что в одном отделе могут работать несколько служащих с фамилией 'Smith'
- Число служащих с фамилией 'Smith' в каждом отделе, где такие служащие работают

```
SELECT EMP.DEPT_NO, COUNT(*)
FROM EMP
WHERE EMP.NAME = 'Smith'
GROUP BY EMP.DEPT_NO;
```

- DISTINCT не требуется, поскольку в запросе содержится раздел GROUP BY, группировка производится в соответствии со значениями столбца EMP.DEPT_NO, и строка результата соответствует одной группе

Общая структура оператора выборки в языке SQL (39)

- Найти номера, имена и номера отделов служащих, родившихся после 15 апреля 1965

```
SELECT EMP.EMP_NO, EMP.EMP_NAME, EMP.DEPT_NO
FROM EMP
WHERE EMP.EMP_BDATE > DATE '1965-04-15';
```

- Дубликатов быть не может, поскольку в список выборки включен столбец, являющийся первичным ключом таблицы EMP
- Найти номера, имена и номера отделов служащих, размер заработной платы которых составляет больше одной десятой объема фонда заработной платы их отделов

```
SELECT EMP.EMP_NO, EMP.EMP_NAME, EMP.DEPT_NO
FROM EMP
WHERE EMP.EMP_SAL > 0.1 *
    (SELECT DEPT_TOTAL_SAL
    FROM DEPT
    WHERE DEPT.DEPT_NO = EMP.DEPT_NO);
```

- Второй операнд операции сравнения содержит подзапрос, возвращающий единственное значение, поскольку логическое выражение раздела WHERE этого подзапроса состоит из условия, однозначно определяющего значение первичного ключа таблицы DEPT
- В условии раздела WHERE подзапроса используется ссылка на столбец таблицы EMP, указанной в разделе FROM «внешнего» запроса
- Подобные подзапросы в терминологии SQL традиционно называются корреляционными

Общая структура оператора выборки в языке SQL (40)

- Эквивалентная формулировка

```
SELECT EMP.EMP_NO, EMP.EMP_NAME, EMP.DEPT_NO
FROM EMP, DEPT
WHERE EMP.DEPT_NO = DEPT.DEPT_NO AND
   EMP.EMP_SAL > 0.1 * DEPT.TOTAL_SAL;
```

- В терминах реляционной алгебры этот запрос представляет собой ограничение (по условию EMP.EMP_SAL > 0.1 * DEPT.TOTAL_SAL) эквисоединения таблиц EMP и DEPT (по условию EMP.DEPT_NO = DEPT.DEPT_NO)
- Подобную операцию часто называют *полусоединением (semijoin)*, поскольку в результирующей таблице используются столбцы только одного из операндов операции эквисоединения
- Мы привели вторую формулировку запроса, преследуя две цели:
  - продемонстрировать, каким образом предикат сравнения можно использовать для задания условия соединения,
  - и показать, что запросы, содержащие вложенные запросы, часто могут быть переформулированы в запросы с соединениями

Общая структура оператора выборки в языке SQL (41)

- Найти номера, имена, номера отделов и имена руководителей отделов служащих, размер заработной платы которых меньше 15000 руб.

```
SELECT EMP1.EMP_NO, EMP1.EMP_NAME,
       EMP1.DEPT_NO, EMP2.EMP_NAME
FROM EMP AS EMP1, EMP AS EMP2, DEPT
WHERE EMP1.EMP_SAL < 15000.00 AND
    EMP1.DEPT_NO = DEPT.DEPT_NO AND
    DEPT.DEPT_MNG = EMP2.EMP_NO;
```

- Эквисоединение ограничения таблицы EMP (по условию EMP_SAL < 15000.00) с таблицами DEPT и EMP (по условиям EMP.DEPT_NO = DEPT.DEPT_NO и DEPT.DEPT_MNG = EMP2.EMP_NO соответственно)
- Таблица EMP участвует в качестве операнда операции эквисоединения два раза
- Поэтому в разделе FROM ей присвоены два псевдонима – EMP1 и EMP2
- Следуя предписанному стандартом порядку выполнения запроса, можно считать, что введение этих псевдонимов обеспечивает переименование столбцов таблицы EMP, требуемое для выполнения раздела FROM с образованием расширенного декартова произведения таблиц-операндов.)
- В данном случае мы имеем дело с полным эквисоединением трех таблиц (а не с полусоединением), поскольку в списке выборки присутствуют имена столбцов каждой из них

Общая структура оператора выборки в языке SQL (42)

- Способ формулировки этого запроса с использованием вложенного подзапроса в качестве элемента списка выборки

```
SELECT EMP.EMP_NO, EMP.EMP_NAME, EMP.DEPT_NO,
    (SELECT EMP_NAME
      FROM EMP
      WHERE EMP_NO = DEPT_MNG)
FROM EMP, DEPT
WHERE EMP.EMP_SAL < 15000.00 AND
    EMP.DEPT_NO = DEPT.DEPT_NO;
```

- В условии выборки подзапроса, участвующего в списке выборки, можно использовать имена столбцов таблиц внешнего запроса

Общая структура оператора выборки в языке SQL (43)

- **Предикат between**
- Предикат позволяет специфицировать условие вхождения в диапазон значений
- Операндами являются строки:

```
between_predicate ::=
    row_value_constructor [ NOT ] BETWEEN
    row_value_constructor AND row_value_constructor
```

- Все три строки-операнды должны иметь одну и ту же степень
- Типы данных соответствующих значений строк-операндов должны быть совместимыми
- Пусть X, Y и Z обозначают первый, второй и третий операнды
- Тогда по определению выражение X NOT BETWEEN Y AND Z эквивалентно выражению NOT (X BETWEEN Y AND Z)
- Выражение X BETWEEN Y AND Z по определению эквивалентно булевскому выражению X >= Y AND X <= Z

Общая структура оператора выборки в языке SQL (44)

- **Примеры запросов с использованием предиката between**
- Найти номера, имена и размер зарплаты служащих, получающих зарплату в размере от 12000 до 15000 руб.

```
SELECT EMP_NO, EMP_NAME, EMP_SAL
FROM EMP
WHERE EMP_SAL BETWEEN 12000.00 AND 15000.00;
```

Общая структура оператора выборки в языке SQL (45)

- Найти номера, имена и размер зарплаты служащих, получающих зарплату, размер которой не меньше средней зарплаты служащих своего отдела и не больше зарплаты руководителя отдела

```
SELECT EMP_NO, EMP_NAME, EMP_SAL
FROM EMP
WHERE EMP_SAL BETWEEN
    (SELECT AVG(EMP1.EMP_SAL)
      FROM EMP EMP1
        WHERE EMP.DEPT_NO = EMP1.DEPT_NO)
          AND
    (SELECT EMP1.EMP_SAL
      FROM EMP EMP1
        WHERE EMP1.EMP_NO =
          (SELECT DEPT.DEPT_MNG
            FROM DEPT
            WHERE DEPT.DEPT_NO = EMP.DEPT_NO));
```

- Диапазон значений предиката BETWEEN задан двумя подзапросами, результатом каждого из которых является единственное значение
  - Первый подзапрос выдает единственное значение, поскольку в списке выборки содержится агрегатная функция (AVG) и отсутствует раздел GROUP BY,
  - а второй – потому что в его разделе WHERE присутствует условие, задающее единственное значение первичного ключа
- В обоих подзапросах таблица EMP получает псевдоним EMP1
  - Поскольку подзапросы выполняются независимо один от другого, использование общего имени не вызывает проблем
- В условии второго подзапроса присутствует более глубоко вложенный подзапрос, и в условии его раздела WHERE используется ссылка на столбец таблицы из самого внешнего раздела FROM

Общая структура оператора выборки в языке SQL (46)

- **Предикат is null**
- Предикат is null позволяет проверить, являются ли неопределенными значения всех элементов строки-операнда:

```
null_predicate ::= row_value_constructor IS [ NOT ] NULL
```

- Пусть X обозначает строку-операнд
- Если значения всех элементов X являются неопределенными, то значением условия X IS NULL является true; иначе – false
- Если ни у одного элемента X значение не является неопределенным, то значением условия X IS NOT NULL является true; иначе – false
  - условие X IS NOT NULL имеет то же значение, что условие NOT X IS NULL для любого X в том и только в том случае, когда степень X равна 1
- Полная семантика предиката null

Вид операнда	Вид условия			
	X IS X NULL	IS NOT NULL	NOT X IS NULL	NOT X IS NOT NULL
Степень 1: значение NULL	true	false	false	true
Степень 1: значение отлично от NULL	false	true	true	false
Степень > 1: у всех элементов значение NULL	true	false	false	true
Степень > 1: у некоторых(не у всех) элементов значение NULL	false	false	true	true
Степень > 1: ни у одного элемента нет значения NULL	false	true	true	false

Общая структура оператора выборки в языке SQL (47)

- **Примеры запросов с использованием предиката null**
- Найти номера, имена и размер зарплаты служащих, получающих зарплату, размер которой не меньше средней зарплаты служащих своего отдела и не больше зарплаты руководителя отдела
  - более понятная формулировка

```
SELECT EMP_NO, EMP_NAME, EMP_SAL
FROM EMP
WHERE DEPT_NO IS NOT NULL AND
    EMP_SAL BETWEEN
    (SELECT AVG(EMP1.EMP_SAL)
      FROM EMP EMP1
      WHERE EMP.DEPT_NO = EMP1.DEPT_NO)
        AND
    (SELECT EMP1.EMP_SAL
      FROM EMP EMP1
      WHERE EMP1.EMP_NO =
        ( SELECT DEPT.DEPT_MNG
        FROM DEPT
        WHERE DEPT.DEPT_NO = EMP.DEPT_NO ) );
```

- Найти номера и имена служащих, номер отдела которых неизвестен

```
SELECT EMP_NO, EMP_NAME
FROM EMP
WHERE DEPT_NO IS NULL;
```

Общая структура оператора выборки в языке SQL (48)

- **Предикат in**
- Предикат позволяет специфицировать условие вхождения строчного значения в указанное множество значений
- Синтаксические правила следующие:

```
in_predicate ::= row_value_constructor [ NOT ]
    IN in_predicate_value
in_predicate_value ::= table_subquery
    | (value_expression_comma_list)
```

- Строка, являющаяся первым операндом, и таблица-второй операнд должны быть одинаковой степени
- Типы данных соответствующих столбцов операндов должны быть совместимы
- Пусть X обозначает строку-первый операнд, а S – множество строк второго операнда
- Обозначим через s строку-элемент этого множества
- Тогда по определению условие X IN S эквивалентно булевскому выражению $OR_{s \in S}(X = s)$
- X IN S принимает значение false в том и только том случае, когда для всех элементов s множества S значением операции сравнения X = s является false
- Иначе значением условия X IN S является unknown
- Для пустого множества S значением X IN S является false
- По определению условие X NOT IN S эквивалентно NOT (X IN S).

Общая структура оператора выборки в языке SQL (49)

- **Примеры запросов с использованием предиката in**
- Найти номера, имена и номера отделов служащих, работающих в отделах 15, 17 и 19

```
SELECT EMP_NO, EMP_NAME, DEPT_NO
FROM EMP
WHERE DEPT_NO IN (15, 17, 19);
```

- Эта формулировка запроса эквивалентна следующей формулировке:

```
SELECT EMP_NO, EMP_NAME, DEPT_NO
FROM EMP
WHERE DEPT_NO = 15
  OR DEPT_NO = 17
  OR DEPT_NO = 19;
```

Общая структура оператора выборки в языке SQL (50)

- Найти номера служащих, не являющихся руководителями отделов и получающих заплату, размер которой равен размеру зарплаты какого-либо руководителя отдела

```
SELECT EMP_NO
FROM EMP
WHERE EMP_NO NOT IN (SELECT DEPT_MNG FROM DEPT)
    AND EMP_SAL IN (SELECT EMP_SAL FROM EMP, DEPT
                        WHERE EMP_NO = DEPT_MNG);
```

- Эквивалентный запрос с соединениями

```
SELECT DISTINCT EMP_NO
FROM EMP, EMP EMP1, DEPT
WHERE EMP_NO NOT IN (SELECT DEPT_MNG FROM DEPT)
 AND EMP_SAL = EMP1_SAL
 AND EMP1.EMP_NO = DEPT.DEPT_MNG;
```

- Мы изменили только ту часть условия, в которой использовался предикат IN, и не затронули предикат NOT IN
  - Запросы с предикатами NOT IN запросами с соединениями так просто не заменяются
- В разделе SELECT было добавлено ключевое слово DISTINCT, потому что
  - в результате запроса во второй формулировке для каждого служащего будет содержаться столько строк, сколько существует руководителей отделов, получающих такую же зарплату, что и данный служащий

Общая структура оператора выборки в языке SQL (51)

- **Предикат like**
- Формально предикат like определяется следующими синтаксическими правилами

```
like_predicate ::= source_value [ NOT ]
    LIKE pattern_value [ ESCAPE escape_value ]
source_value ::= value_expression
pattern_value ::= value_expression
escape_value ::= value_expression
```

- Все три операнда (source_value, pattern_value и escape_value) должны быть одного типа: либо типа символьных строк, либо типа битовых строк
- В первом случае значением последнего операнда должна быть строка из одного символа, во втором – строка из 8 бит
- Второй операнд, как правило, задается литералом соответствующего типа
- В обоих случаях значение предиката равняется true в том и только в том случае, когда исходная строка (source_value) может быть сопоставлена с заданным шаблоном (pattern_value)

Общая структура оператора выборки в языке SQL (52)

- Если обрабатываются символьные строки, и если раздел ESCAPE условия отсутствует, то при сопоставлении шаблона со строкой производится специальная интерпретация двух символов шаблона:
  - символ подчеркивания ('_') обозначает любой одиночный символ;
  - символ процента ('%') обозначает последовательность произвольных символов произвольной длины (длина последовательности может быть нулевой)
- Если же раздел ESCAPE присутствует и специфицирует некоторый одиночный символ x, то пары символов «x_» и «x%» представляют одиночные символы «_» и «%» соответственно
- В случае обработки битовых строк сопоставление шаблона со строкой производится восьмерками соседних бит (*октетами*)
  - При сопоставлении шаблона со строкой производится специальная интерпретация октетов со значениями X'25' и X'5F' (коды символов подчеркивания и процента в кодировке ASCII)
  - Первый октет обозначает любой одиночный октет, а второй – последовательность произвольной длины произвольных октетов (длина может быть нулевой)
- В разделе ESCAPE указывается октет, отменяющий специальную интерпретацию октетов X'25' и X'5F‘
- Значение предиката like есть unknown, если значение первого или второго операндов является неопределенным
- Условие x NOT LIKE y ESCAPE z эквивалентно условию NOT x LIKE y ESCAPE z

Общая структура оператора выборки в языке SQL (53)

- **Примеры запросов с использованием предиката like**
- Найти номера проектов, в названии которых присутствуют слова 'next' и 'step'. Слова должны следовать именно в такой последовательности, но слово 'next' может быть первым в названии проекта

```
SELECT PRO_TITLE
FROM PRO
WHERE PRO_TITLE LIKE '%next%step%'
    OR PRO_TITLE LIKE 'Next%step%';
```

- Выполнение запроса, скорее всего, вынудит СУБД просмотреть все строки таблицы PRO и для каждой строки выполнить две проверки столбца PRO_TITLE
- Можно немного улучшить формулировку с небольшим риском получить неверный ответ

```
SELECT PRO_TITLE
FROM PRO
WHERE PRO_TITLE LIKE '%ext%step%';
```

Общая структура оператора выборки в языке SQL (54)

- Найти номера отделов, служащие которых являются менеджерами проектов, и название каждого из этих проектов начинается с названия отдела

```
SELECT DISTINCT DEPT.DEPT_NO
FROM EMP, DEPT, PRO
WHERE EMP.EMP_NO = PRO.PRO_MNG
 AND EMP.DEPT_NO = DEPT.DEPT_NO
 AND PRO.PRO_TITLE LIKE DEPT.DEPT_NAME || '%';
```

- Вот как может выглядеть формулировка этого запроса, если использовать вложенные подзапросы

```
SELECT DEPT.DEPT_NO
FROM DEPT
WHERE DEPT.DEPT_NO IN
    (SELECT EMP.DEPT_NO
    FROM EMP
    WHERE EMP.EMP_NO IN
      (SELECT PRO.PRO_MNG FROM PRO
    WHERE PRO.PRO_TITLE LIKE DEPT.DEPT_NAME || '%'));
```

Общая структура оператора выборки в языке SQL (55)

- **Предикат similar**
- Формально предикат similar определяется следующими синтаксическими правилами:

```
similar_predicate ::= source_value [ NOT ]
    SIMILAR TO pattern_value [ ESCAPE escape_value ]
source_value ::= character_expression
pattern_value ::= character_expression
escape_value ::= character_expression
```

- Все три операнда (source_value, pattern_value и escape_value) должны иметь тип символьных строк
- Значением последнего операнда должна быть строка из одного символа
- Второй операнд, как правило, задается литералом соответствующего типа
- В обоих случаях значение предиката равняется true в том и только в том случае, когда шаблон (pattern_value) должным образом сопоставляется с исходной строкой (source_value)
- Основное отличие предиката similar от рассмотренного ранее предиката like состоит в существенно расширенных возможностях задания шаблона

Общая структура оператора выборки в языке SQL (56)

- Регулярные выражения предиката similar определяются следующими синтаксическими правилами:

```
regular_expression ::= regular_term
    | regular_expression vertical_bar regular_term
regular_term ::= regular_factor
    | regular_term regular_factor
regular_factor ::= regular_primary
    | regular_primary *
    | regular_primary +
regular_primary ::= character_specifier
    | %
    | regular_character_set
    | ( regular_expression )
character_specifier ::= non_escape_character
    | escape_character
regular_character_set ::= _
    | left_bracket
        character_enumeration_list right_bracket
    | left_bracket
        ^ character_enumeration_list right_bracket
    | left_bracket : regular_charset_id : right_bracket
character_enumeration ::= character_specifier
    | character_specifier - character_specifier
regular_charset_id ::= ALPHA | UPPER | LOWER
    | DIGIT | ALNUM
```

- Поскольку в синтаксических правилах регулярных выражений символы «|», «[» и «]», используемые нами в качестве метасимволов в BNF, являются терминальными символами, они изображены как vertical_bar, left_bracket и right_bracket соответственно

Общая структура оператора выборки в языке SQL (57)

- Создаваемое по приведенным правилам регулярное выражение представляет собой символьную строку, содержащую все символы, которые требуется явно сопоставлять с символами строки-источника
- В строке могут находиться специальные символы, представляющие собой заменители обычных символов («%» и «_»), обозначения операций («|»), показатели числа возможных повторений («*» и «+») и т. д.
- При вычислении регулярного выражения образуются все возможные символьные строки, не содержащие специальных символов и соответствующие исходному шаблону
- Тем самым, значением предиката similar является true в том и только в том случае, когда среди всех символьных строк, генерируемых по регулярному выражению pattern_value, найдется символьная строка, совпадающая с source_value
- Рассмотрим несколько примеров регулярных выражений
- Выражение '(This is string1)|(This is string2)' производит две символьные строки: '(This is string1)' и '(This is string2)‘
- В общем случае в круглых скобках могут находиться произвольные регулярные выражения rexp1 и rexp2
- Результатом вычисления '(rexp1)|(rexp2)' является множество символьных строк, генерируемых выражением rexp1, объединенное с множеством символьных строк, генерируемых выражением rexp2

Общая структура оператора выборки в языке SQL (58)

- Выражение 'This is string [12]*' генерирует символьные строки 'This is string ', 'This is string 1', 'This is string 2', 'This is string 11', 'This is string 22', 'This is string 12', 'This is string 22', 'This is string 111' и т. д.
- Конструкция в квадратных скобках представляет собой один из вариантов определения набора символов (regular_character_set)
- В данном случае символы, входящие в определяемый набор, просто перечисляются
- При вычислении регулярного выражения в каждой из генерируемых символьных строк конструкция в квадратных скобках заменяется одним из символов соответствующего набора
- Специальный символ «*», стоящий после закрывающей квадратной скобки, является показателем числа повторений
- «Звездочка» означает, что в генерируемых символьных строках элемент регулярного выражения, непосредственно предшествующий «звездочке», может появляться ноль или более раз
- Использование в такой же ситуации специального символа «+» означает, что в генерируемых символьных строках элемент регулярного выражения, непосредственно предшествующий символу «плюс», может появляться один или более раз.

Общая структура оператора выборки в языке SQL (59)

- Другая форма определения набора символов иллюстрируется регулярным выражением 'This is string [:DIGIT:]‘
- В этом случае конструкция в квадратных скобках представляет любой одиночный символ, изображающий десятичную цифру
- Другими допустимыми в SQL идентификаторами наборов символов (regular_charset_id) являются ALPHA (любой символ алфавита), UPPER (любой символ верхнего регистра), LOWER (любой символ нижнего регистра) и ALNUM (любой алфавитно-цифровой символ)
- Определяемый набор символов может задаваться нижней и верхней границей диапазона значений кодов допустимых символов
- Например, в регулярном выражении 'This is string [3-8]' конструкция в квадратных скобках представляет собой любой одиночный символ, изображающий цифры от 3 до 8 включительно
- Заметим, что при задании диапазона можно использовать любые символы, но требуется, чтобы значение кода символа левой границы диапазона было не больше значения кода символа правой границы
- Наконец, имеется еще одна возможность определения набора символов
- Соответствующая конструкция позволяет указать, какие символы из общего набора символов SQL не входят в определяемый набор символов
- Например, регулярное выражение '_S[^t]*ing%' генерирует все символьные строки, у которых вторым символом является «S», за которым (не обязательно непосредственно) следует подстрока «ing», но между «S» и «ing» отсутствуют вхождения символа «t»
- Как и в предикате like, символ, определенный в разделе ESCAPE, поставленный перед любым специальным символом, отменяет специальную интерпретацию этого символа.

Общая структура оператора выборки в языке SQL (60)

- **Примеры запросов с использованием предиката similar**
- Найти номера и названия отделов, название которых начинается со слов 'Hardware' или 'Software', а за ними (не обязательно непосредственно) следует последовательность десятичных цифр, предваряемых символом подчеркивания

```
SELECT DEPT_NAME, DEPT_NO
FROM DEPT
WHERE DEPT_NAME SIMILAR TO
    '(HARD|SOFT)WARE%\_[:DIGIT:]+' ESCAPE '\';
```

- Найти номера и названия проектов, название которых не начинается с последовательности цифр

```
SELECT DEPT_NAME, DEPT_NO
FROM DEPT
WHERE DEPT_NAME SIMILAR TO '[^1-9]+%';
```

Общая структура оператора выборки в языке SQL (61)

- **Предикат exists**
- Предикат exists определяется следующим синтаксическим правилом:

```
exists_predicate ::= EXISTS (query_expression)
```

- Значением условия EXISTS (query_expression) является true в том и только в том случае, когда мощность таблицы-результата выражения запросов больше нуля, иначе значением условия является false
- **Примеры запросов с использованием предиката exists**
- Найти номера отделов, среди служащих которых имеются менеджеры проектов

```
SELECT DEPT.DEPT_NO
FROM DEPT
WHERE EXISTS
    (SELECT EMP.EMP_NO
    FROM EMP
    WHERE EMP.DEPT_NO = DEPT.DEPT_NO
        AND EXISTS
            (SELECT PRO.PRO_MNG
            FROM PRO
            WHERE PRO.PRO_MNG = EMP.EMP_NO));
```

Общая структура оператора выборки в языке SQL (62)

- Формулировку можно упростить, избавившись от самого вложенного запроса

```
SELECT DEPT.DEPT_NO
FROM DEPT
WHERE EXISTS
    (SELECT EMP.EMP_NO
      FROM EMP, PRO
      WHERE EMP.DEPT_NO = DEPT.DEPT_NO
        AND PRO.PRO_MNG = EMP.EMP_NO);
```

- По смыслу предиката EXISTS список выборки во вложенном подзапросе является несущественным, и формулировку запроса можно изменить, например, следующим образом

```
SELECT DEPT.DEPT_NO
FROM DEPT
WHERE EXISTS
    (SELECT *
      FROM EMP, DEPT
      WHERE EMP.DEPT_NO = DEPT.DEPT_NO
        AND PRO.PRO_MNG = EMP.EMP_NO);
```

Общая структура оператора выборки в языке SQL (63)

- Запросы с предикатом EXISTS можно также переформулировать в виде запросов с предикатом сравнения

```
SELECT DEPT.DEPT_NO
FROM DEPT
WHERE (SELECT COUNT(*)
    FROM EMP, DEPT
    WHERE EMP.DEPT_NO = DEPT.DEPT_NO
     AND PRO.PRO_MNG = EMP.EMP_NO ) >= 1;
```

- Найти номера отделов, размер заработной платы служащих которых не превышает размер заработной платы руководителя отдела

```
SELECT DEPT.DEPT_NO
FROM DEPT
WHERE NOT EXISTS
    (SELECT *
      FROM EMP EMP1, EMP EMP2
      WHERE EMP1.EMP_NO = DEPT.DEPT_MNG AND
        EMP2.DEPT_NO = DEPT.DEPT_NO AND
        EMP2.EMP_SAL > EMP1.EMP_SAL);
```

Общая структура оператора выборки в языке SQL (64)

- **Предикат сравнения с квантором**
- Этот предикат позволяет специфицировать квантифицированное сравнение строчного значения и определяется следующим синтаксическим правилом:

```
quantified_comparison_predicate ::= row_value_constructor
    comp_op { ALL | SOME | ANY } query_expression
```

- Степень первого операнда должна быть такой же, как и степень таблицы-результата выражения запросов
- Типы данных значений строки-операнда должны быть совместимы с типами данных соответствующих столбцов выражения запроса
- Сравнение строк производится по тем же правилам, что и для предиката сравнения.

Общая структура оператора выборки в языке SQL (65)

- **Предикат сравнения с квантором**
- Обозначим через x строку-первый операнд, а через S – результат вычисления выражения запроса
- Пусть s обозначает произвольную строку таблицы S
- Тогда:
- условие x comp_op ALL S имеет значение true в том и только в том случае, когда S пусто, или значение условия x comp_op s равно true *для каждой строки* s, входящей в S
- условие x comp_op ALL S имеет значение false в том и только в том случае, когда значение предиката x comp_op s равно false *хотя бы для одной строки* s, входящей в S
- В остальных случаях значение условия x comp_op ALL S равно unknown
- условие x comp_op SOME S имеет значение false в том и только в том случае, когда S пусто, или значение условия x comp_op s равно false *для каждой строки* s, входящей в S
- условие x comp_op SOME S имеет значение true в том и только в том случае, когда значение предиката x comp_op s равно true *хотя бы для одной* строки s, входящей в S
- в остальных случаях значение условия x comp_op SOME S равно unknown;
- условие x comp_op ANY S эквивалентно условию x comp_op SOME S.

Общая структура оператора выборки в языке SQL (66)

- **Примеры запросов с использованием предиката сравнения с квантором**
- Найти номера служащих отдела номер 65, зарплата которых в этом отделе не является минимальной

```
SELECT EMP_NO
FROM EMP
WHERE DEPT_NO = 65
    AND EMP_SAL > SOME (SELECT EMP1.EMP_SAL
      FROM EMP EMP1
      WHERE EMP.DEPT_NO = EMP1.DEPT_NO);
```

- Одна из возможных альтернативных формулировок этого запроса может основываться на использовании предиката EXISTS

```
SELECT EMP_NO
FROM EMP
WHERE DEPT_NO = 65
    AND EXISTS(SELECT *
      FROM EMP EMP1
      WHERE EMP.DEPT_NO = EMP1.DEPT_NO
        AND EMP.EMP_SAL > EMP1.EMP_SAL);
```

- Альтернативная формулировка этого запроса, основанная на использовании агрегатной функции MIN

```
SELECT EMP_NO
FROM EMP
WHERE DEPT_NO = 65 AND
    EMP_SAL > (SELECT MIN(EMP1.EMP_SAL)
      FROM EMP EMP1
      WHERE EMP.DEPT_NO = EMP1.DEPT_NO);
```

Общая структура оператора выборки в языке SQL (67)

- Найти номера и имена служащих отдела 65, однофамильцы которых работают в этом же отделе

```
SELECT EMP_NO, EMP_NAME
FROM EMP
WHERE DEPT_NO = 65 AND
    EMP_NAME = SOME (SELECT EMP1.EMP_NAME
      FROM EMP EMP1
      WHERE EMP.DEPT_NO = EMP1.DEPT_NO
        AND EMP.EMP_NO <> EMP1.EMP_NO);
```

- Эта формулировка эквивалентна следующей формулировке

```
SELECT EMP_NO, EMP_NAME
FROM EMP
WHERE DEPT_NO = 65 AND
    EMP_NAME IN (SELECT EMP1.EMP_NAME
      FROM EMP EMP1
      WHERE EMP.DEPT_NO = EMP1.DEPT_NO
        AND EMP.EMP_NO <> EMP1.EMP_NO);
```

Общая структура оператора выборки в языке SQL (68)

- Возможна формулировка с использованием агрегатной функции COUNT

```
SELECT EMP_NO, EMP_NAME
FROM EMP
WHERE DEPT_NO = 65 AND
    (SELECT COUNT(*)
      FROM EMP EMP1
      WHERE EMP.DEPT_NO = EMP1.DEPT_NO
        AND EMP.EMP_NO <> EMP1.EMP_NO ) >= 1;
```

- Наиболее лаконичным образом этот запрос можно сформулировать с использованием соединения

```
SELECT DISTINCT EMP.EMP_NO, EMP.EMP_NAME
FROM EMP, EMP EMP1
WHERE EMP.DEPT_NO = 65
    AND EMP.EMP_NAME = EMP1.EMP_NAME
    AND EMP.DEPT_NO = EMP1.DEPT_NO
    AND EMP.EMP_NO <> EMP1.EMP_NO;
```

Общая структура оператора выборки в языке SQL (69)

- Найти номера служащих отдела номер 65, зарплата которых в этом отделе является максимальной

```
SELECT EMP_NO
FROM EMP
WHERE DEPT_NO = 65
 AND EMP_SAL >= ALL(SELECT EMP1.EMP_SAL
    FROM EMP EMP1
    WHERE EMP.DEPT_NO = EMP1.DEPT_NO);
```

- Одна из возможных альтернативных формулировок этого запроса может основываться на использовании предиката NOT EXISTS

```
SELECT EMP_NO
FROM EMP
WHERE DEPT_NO = 65
 AND NOT EXISTS (SELECT *
    FROM EMP EMP1
    WHERE EMP.DEPT_NO = EMP1.DEPT_NO
    AND EMP.EMP_SAL < EMP1.EMP_SAL);
```

Общая структура оператора выборки в языке SQL (70)

- Можно сформулировать этот же запрос с использованием агрегатной функции MAX

```
SELECT EMP_NO
FROM EMP
WHERE DEPT_NO = 65
 AND EMP_SAL = (SELECT MAX(EMP1.EMP_SAL)
    FROM EMP EMP1
    WHERE EMP.DEPT_NO = EMP1.DEPT_NO);
```

- Найти номера и имена служащих, не имеющих однофамильцев

```
SELECT EMP_NO, EMP_NAME
FROM EMP
WHERE EMP_NAME <> ALL (SELECT EMP1.EMP_NAME
    FROM EMP EMP1
    WHERE EMP1.EMP_NO <> EMP.EMP_NO);
```

- Формулировка в виде запроса с соединением здесь не проходит