

## Лекция 8: Агрегирование и групповые функции

Агрегирующие функции позволяют получать из таблицы сводную (агрегированную) информацию, выполняя операции над группой строк таблицы. Для задания в **SELECT**-запросе агрегирующих операций используются следующие ключевые слова:

- **COUNT** определяет количество строк или значений поля, выбранных посредством запроса, и не являющихся **NULL**-значениями;
- **SUM** – вычисляет арифметическую сумму всех выбранных значений данного поля;
- **AVG** вычисляет среднее значение для всех выбранных значений данного поля;
- **MAX** вычисляет наибольшее из всех выбранных значений данного поля;
- **MIN** вычисляет наименьшее из всех выбранных значений данного поля.

В **SELECT**-запросе агрегирующие функции используются аналогично именам полей, при этом последние (имена полей) используются в качестве аргументов этих функций.

Функция **AVG** предназначена для подсчета среднего значения поля на множестве записей таблицы.

Например, для определения среднего значения поля MARK (оценки) по всем записям таблицы EXAM\_MARKS можно использовать запрос с функцией **AVG** следующего вида:

```
SELECT AVG(MARK)  
FROM EXAM_MARKS;
```

Для подсчета общего количества строк в таблице следует использовать функцию **COUNT** со звездочкой.

```
SELECT COUNT(*)  
FROM EXAM_MARKS;
```

Аргументы **DISTINCT** и **ALL** позволяют, соответственно, исключать и включать дубликаты обрабатываемых функцией **COUNT** значений, при этом необходимо учитывать, что при использовании опции **ALL** значения **NULL** все равно не войдут в число подсчитываемых значений.

```
SELECT COUNT(DISTINCT SUBJ_ID)  
FROM SUBJECT;
```

Предложение **GROUP BY** (ГРУППИРОВАТЬ ПО) позволяет группировать записи в подмножества, определяемые значениями какого-либо поля, и применять агрегирующие функции уже не ко всем записям таблицы, а отдельно к каждой сформированной группе.

Предположим, требуется найти максимальное значение оценки, полученной каждым студентом. Запрос будет выглядеть следующим образом:

```
SELECT STUDENT_ID, MAX(MARK)
FROM EXAM_MARKS
GROUP BY STUDENT_ID;
```

Выбираемые из таблицы EXAM\_MARKS записи группируются по значениям поля STUDENT\_ID, указанного в предложении **GROUP BY**, и для каждой группы находится максимальное значение поля MARK. Предложение **GROUP BY** позволяет применять агрегирующие функции к каждой группе, определяемой общим значением поля (или полей), указанных в этом предложении. В приведенном запросе рассматриваются группы записей, сгруппированные по идентификаторам студентов.

В конструкции **GROUP BY** для группирования может быть использовано более одного столбца. Например:

```
SELECT STUDENT_ID, SUBJ_ID, MAX(MARK)
FROM EXAM_MARKS
GROUP BY STUDENT_ID, SUBJ_ID;
```

В этом случае строки вначале группируются по значениям первого столбца, а внутри этих групп – в подгруппы по значениям второго столбца. Таким образом, **GROUP BY** не только устанавливает столбцы, по которым осуществляется группирование, но и указывает порядок разбиения столбцов на группы.

Следует иметь в виду, что в предложении **GROUP BY** должны быть указаны все выбираемые столбцы, приведенные после ключевого слова **SELECT**, кроме столбцов, указанных в качестве аргумента в агрегирующей функции.

При необходимости часть сформированных с помощью **GROUP BY** групп может быть исключена с помощью предложения **HAVING**.

Предложение **HAVING** определяет критерий, по которому группы следует включать в выходные данные, по аналогии с предложением **WHERE**, которое осуществляет это для отдельных строк.

```
SELECT SUBJ_NAME, MAX(HOUR)
FROM SUBJECT
GROUP BY SUBJ_NAME
HAVING MAX(HOUR) >= 72;
```

В условии, задаваемом предложением **HAVING**, указывают только поля или выражения, которые на выходе имеют единственное значение для каждой выводимой группы.

## Вложенные подзапросы

SQL позволяет использовать одни запросы внутри других запросов, то есть вкладывать запросы друг в друга. Предположим, известна фамилия студента (“Петров”), но неизвестно значение поля `STUDENT_ID` для него. Чтобы извлечь данные обо всех оценках этого студента, можно записать следующий запрос:

```
SELECT *
FROM EXAM_MARKS
WHERE STUDENT_ID =
  ( SELECT STUDENT_ID
    FROM STUDENT SURNAME = 'Петров');
```

Как работает запрос SQL со связанным подзапросом?

- Выбирается строка из таблицы, имя которой указано во внешнем запросе.
- Выполняется подзапрос и полученное в результате его выполнения значение применяется для анализа этой строки в условии предложения **WHERE** внешнего запроса.
- По результату оценки этого условия принимается решение о включении или не включении строки в состав выходных данных.
- Процедура повторяется для следующей строки таблицы внешнего запроса.

Следует обратить внимание, что приведенный выше запрос корректен только в том случае, если в результате выполнения указанного в скобках подзапроса возвращается *единственное значение*. Если в результате выполнения подзапроса будет возвращено несколько значений, то этот подзапрос будет ошибочным. В данном примере это произойдет, если в таблице STUDENT будет несколько записей со значениями поля SURNAME = 'Петров'.

В некоторых случаях для гарантии получения единственного значения в результате выполнения подзапроса используется **DISTINCT**. Одним из видов функций, которые автоматически *всегда* выдают в результате единственное значение для любого количества строк, являются агрегирующие функции.

Оператор **IN** также широко применяется в подзапросах. Он задает список значений, с которыми сравниваются другие значения для определения истинности задаваемого этим оператором предиката.

Данные обо всех оценках (таблица EXAM\_MARKS) студентов из Воронежа можно выбрать с помощью следующего запроса:

```
SELECT *  
  FROM EXAM_MARKS  
 WHERE STUDENT_ID IN  
      ( SELECT STUDENT_ID  
        FROM STUDENT  
        WHERE CITY = 'Воронеж');
```

## Формирование связанных подзапросов

При использовании подзапросов во внутреннем запросе можно ссылаться на таблицу, имя которой указано в предложении **FROM** внешнего запроса. В этом случае такой *связанный* подзапрос выполняется по одному разу для *каждой* строки таблицы основного запроса.

**Пример:** выбрать сведения обо всех предметах обучения, по которым проводился экзамен 20 января 1999 г.

```
SELECT *  
  FROM SUBJECT SU  
 WHERE '20/01/1999' IN  
      ( SELECT EXAM_DATE  
        FROM EXAM_MARKS EX  
        WHERE SU.SUBJ_ID = EX.SUBJ_ID);
```

В некоторых СУБД для выполнения этого запроса, возможно, потребуется преобразование значения даты в символьный тип. В приведенном запросе SU и EX являются псевдонимами (алиасами), то есть специально вводимыми именами, которые могут быть использованы в данном запросе вместо настоящих имен. В приведенном примере они используются вместо имен таблиц SUBJECT и EXAM\_MARKS. .

Эту же задачу можно решить с помощью операции соединения таблиц:

```
SELECT DISTINCT SU.SUBJ_ID, SUBJ_NAME, HOUR, SEMESTER  
  FROM SUBJECT FIRST, EXAM_MARKS SECOND  
 WHERE FIRST.SUBJ_ID = SECOND.SUBJ_ID  
 AND SECOND.EXAM_DATE = '20/01/1999';
```

В этом выражении алиасами таблиц являются имена FIRST и SECOND.

Можно использовать подзапросы, связывающие таблицу со своей собственной копией. Например, надо найти идентификаторы, фамилии и стипендии студентов, получающих стипендию выше средней на курсе, на котором они учатся.

```
SELECT DISTINCT STUDENT_ID, SURNAME, STIPEND  
  FROM STUDENT E1  
 WHERE STIPEND >  
      ( SELECT AVG(STIPEND)  
        FROM STUDENT E2
```

**WHERE** E1.KURS = E2.KURS);

Тот же результат можно получить с помощью следующего запроса:

```
SELECT DISTINCT STUDENT_ID, SURNAME, STIPEND
FROM STUDENT E1,
  (SELECT KURS, AVG(STIPEND) AS AVG_STIPEND
   FROM STUDENT E2
   GROUP BY E2.KURS) E3
WHERE E1.STIPEND > AVG_STIPEND AND E1.KURS=E3.KURS;
```

Обратите внимание – второй запрос будет выполнен гораздо быстрее. Дело в том, что в первом варианте запроса агрегирующая функция **AVG** выполняется над таблицей, указанной в подзапросе, для *каждой* строки внешнего запроса. В другом варианте вторая таблица (алиас E2) обрабатывается агрегирующей функцией один раз, в результате чего формируется вспомогательная таблица (в запросе она имеет алиас E3), со строками которой затем соединяются строки первой таблицы (алиас E1). Следует иметь в виду, что реальное время выполнения запроса в большой степени зависит от оптимизатора запросов конкретной СУБД.

## Связанные подзапросы в HAVING

В разделе указывалось, что предложение **GROUP BY** позволяет группировать выводимые **SELECT**-запросом записи по значению некоторого поля. Использование предложения **HAVING** позволяет при выводе осуществлять фильтрацию таких групп. Предикат предложения **HAVING** оценивается не для каждой строки результата, а для каждой группы выходных записей, сформированной предложением **GROUP BY** внешнего запроса.

Пусть, например, необходимо по данным из таблицы EXAM\_MARKS определить сумму полученных студентами оценок (значений поля MARK), сгруппировав значения оценок по датам экзаменов и исключив те дни, когда число студентов, сдававших в течение дня экзамены, было меньше 10.

```
SELECT EXAM_DATE, SUM(MARK)
FROM EXAM_MARKS A
GROUP BY EXAM_DATE
```

```

HAVING 10 <
    ( SELECT COUNT(MARK)
      FROM EXAM_MARKS B
      WHERE A.EXAM_DATE = B.EXAM_DATE);

```

Подзапрос вычисляет количество строк с одной и той же датой, совпадающей с датой, для которой сформирована очередная группа основного запроса.

## Использование оператора EXISTS

Используемый в SQL оператор **EXISTS** (СУЩЕСТВУЕТ) генерирует значение **ИСТИНА** или **ЛОЖЬ**, подобно булеву выражению. Используя подзапросы в качестве аргумента, этот оператор оценивает результат выполнения подзапроса как истинный, если этот подзапрос генерирует выходные данные, то есть в случае *существования* (возврата) хотя бы одного найденного значения. В противном случае результат подзапроса – ложный. Оператор **EXISTS** не может принимать значение **unknown** (неизвестно).

Пусть, например, нужно извлечь из таблицы EXAM\_MARKS данные о студентах, получивших хотя бы одну неудовлетворительную оценку.

```

SELECT DISTINCT STUDENT_ID
FROM EXAM_MARKS A
WHERE EXISTS
    ( SELECT *
      FROM EXAM_MARKS B
      WHERE MARK < 3
        AND B.STUDENT_ID=A.STUDENT_ID);

```

При использовании связанных подзапросов предложение **EXISTS** анализирует каждую строку таблицы, на которую имеется ссылка во внешнем запросе. Главный запрос получает строки-кандидаты на проверку условия. Для каждой строки-кандидата выполняется подзапрос. Как только подзапрос находит строку, где в столбце MARK значение удовлетворяет условию, он прекращает выполнение и возвращает значение **ИСТИНА** внешнему запросу, который затем анализирует свою строку-кандидата.

Например, требуется получить идентификаторы предметов обучения, экзамены по которым сдавались не одним, а несколькими студентами:

```

SELECT DISTINCT SUBJ_ID
FROM EXAM_MARKS A
WHERE EXISTS
    ( SELECT *
      FROM EXAM_MARKS B
      WHERE A.SUBJ_ID = B.SUBJ_ID
      AND A.STUDENT_ID <> B.STUDENT_ID);

```

Часто **EXISTS** применяется с оператором **NOT** (по-русски **NOT EXISTS** интерпретируется, как “*не существует ...*”). Если предыдущий запрос сформулировать следующим образом – найти идентификаторы предметов обучения, которые сдавались одним и только одним студентом (другими словами, для которых не существует другого сдававшего студента), то достаточно просто поставить **NOT** перед **EXISTS**.

Следует иметь в виду, что в подзапросе, указываемом в операторе **EXISTS**, *нельзя использовать агрегирующие функции*.

Возможности применения вложенных запросов весьма разнообразны. Например, пусть из таблицы **STUDENT** требуется извлечь строки для каждого студента, сдавшего более одного предмета.

```

SELECT *
FROM STUDENT FIRST
WHERE EXISTS
    (SELECT SUBJ_ID
     FROM EXAM_MARKS SECOND
     GROUP BY SUBJ_ID
     HAVING COUNT(SUBJ_ID) >1
     WHERE FIRST.STUDENT_ID = SECOND.STUDENT_ID);

```

## Оператор объединения UNION

Оператор **UNION** используется для объединения выходных данных двух или более SQL-запросов в единое множество строк и столбцов. Например, для того, чтобы получить в одной таблице фамилии и идентификаторы студентов и преподавателей из Москвы, можно использовать следующий запрос.

```
SELECT 'Студент_____', SURNAME, STUDENT_ID
FROM STUDENT
WHERE CITY = 'Москва'
UNION
SELECT 'Преподаватель', SURNAME, LECTURER_ID
FROM LECTURER
WHERE CITY = 'Москва';
```

Обратите внимание на то, что символом “;” (точка с запятой) оканчивается только последний запрос. Отсутствие этого символа в конце **SELECT**-запроса означает, что следующий за ним запрос также, как и он сам, является частью общего запроса с **UNION**.

Использование оператора **UNION** возможно только при объединении запросов, соответствующие столбцы которых *совместимы по объединению*. То есть, соответствующие числовые поля должны иметь полностью совпадающие тип и размер, символьные поля должны иметь точно совпадающее количество символов. Если **NULL**-значения запрещены для столбца хотя бы одного любого подзапроса объединения, то они должны быть запрещены и для всех соответствующих столбцов в других подзапросах объединения.

## Внешнее объединение

Часто полезна операция объединения двух запросов, в которой второй запрос выбирает строки, исключенные первым. Такая операция называется внешним объединением.

Рассмотрим пример. Пусть в таблице **STUDENT** имеются записи о студентах, в которых не указан идентификатор университета. Требуется составить список студентов с указанием наименования университета для тех студентов, у которых эти данные есть, но при этом не отбрасывая и студентов, у которых университет не указан. Можно получить желаемые сведения, сформировав объединение двух запросов, один из которых выполняет выборку студентов с названиями их университетов, а второй выбирает студентов с **NULL**-значениями в поле **UNIV\_ID**. В данном случае оказывается полезной возможность вставки в запрос констант, в нашем случае текстовой константы ‘не известен’, чтобы отметить в списке тех студентов, у которых отсутствует информация об университете.

```

SELECT SURNAME, NAME, UNIV_NAME
FROM STUDENT, UNIVERSITY
WHERE STUDENT.UNIV_ID = UNIVERSITY.UNIV_ID
UNION
SELECT SURNAME, NAME, 'не известен'
FROM STUDENT
WHERE UNIV_ID IS NULL
ORDER BY 1;

```

Для совместимости столбцов объединяемых запросов константу 'не известен' во втором запросе следует дополнить пробелами так, чтобы ее длина соответствовала длине поля UNIV\_NAME или использовать для согласования типов функцию **CAST**. В некоторых СУБД согласование типов поля и замещающей его текстовой константы осуществляется автоматически.

## Соединение таблиц с использованием оператора JOIN

Если в операторе **SELECT** после ключевого слова **FROM** указывается не одна, а две таблицы, то в результате выполнения запроса, в котором отсутствует предложение **WHERE**, каждая строка одной таблицы будет соединена с каждой строкой второй таблицы. Такая операция называется *декартовым произведением* или *полным (CROSS) соединением* таблиц базы данных. Сама по себе эта операция не имеет практического значения, более того, при ошибочном использовании она может привести к неожиданным нештатным ситуациям, так как в этом случае в ответе на запрос количество записей будет равно произведению числа записей в соединяемых таблицах, то есть может оказаться чрезвычайно большим. Соединение таблиц имеет смысл тогда, когда соединяются *не все* строки исходных таблиц, а только те, которые интересуют пользователя. Такое ограничение может быть осуществлено с помощью использования в запросе соответствующего условия в предложении **WHERE**. Таким образом, SQL позволяет выводить информацию из нескольких таблиц, связывая их по значениям определенных полей.

Например, если необходимо получить фамилии студентов (таблица STUDENT) и для каждого студента – названия университетов (таблица

UNIVERSITY), расположенных в городе, где живет студент, то необходимо получить все комбинации записей о студентах и университетах в обеих таблицах, в которых значение поля CITY совпадает. Это можно сделать с помощью следующего запроса.

```
SELECT STUDENT.SURNAME, UNIVERSITY.UNIV_NAME, STUDENT.CITY  
FROM STUDENT, UNIVERSITY  
WHERE STUDENT.CITY = UNIVERSITY.CITY;
```

Соединение, использующее предикаты, основанные на равенствах, называется *эквисоединением*. Рассмотренный пример соединения таблиц относится к виду так называемого *внутреннего (INNER) соединения*. При таком типе соединения соединяются только те строки таблиц, для которых является истинным предикат, задаваемый в предложении **ON** выполняемого запроса.

Приведенный выше запрос может быть записан иначе, с использованием ключевого слова **JOIN**.

```
SELECT STUDENT.SURNAME, UNIVERSITY.UNIV_NAME, STUDENT.CITY  
FROM STUDENT INNER JOIN UNIVERSITY  
ON STUDENT.CITY = UNIVERSITY.CITY;
```

Ключевое слово **INNER** в запросе может быть опущено, так как эта опция в операторе **JOIN** действует по умолчанию.

Рассмотренный выше случай полного соединения (декартова произведения таблиц) с использованием ключевого слова **JOIN** будет выглядеть следующим образом

```
SELECT * FROM STUDENT JOIN UNIVERSITY;
```

UNIVERSITY), расположенных в городе, где живет студент, то необходимо получить все комбинации записей о студентах и университетах в обеих таблицах, в которых значение поля CITY совпадает. Это можно сделать с помощью следующего запроса.

```
SELECT STUDENT.SURNAME, UNIVERSITY.UNIV_NAME, STUDENT.CITY  
FROM STUDENT, UNIVERSITY  
WHERE STUDENT.CITY = UNIVERSITY.CITY;
```

Соединение, использующее предикаты, основанные на равенствах, называется *эквисоединением*. Рассмотренный пример соединения таблиц относится к виду так называемого *внутреннего (INNER) соединения*. При таком типе соединения соединяются только те строки таблиц, для которых является истинным предикат, задаваемый в предложении **ON** выполняемого запроса.

Приведенный выше запрос может быть записан иначе, с использованием ключевого слова **JOIN**.

```
SELECT STUDENT.SURNAME, UNIVERSITY.UNIV_NAME, STUDENT.CITY  
FROM STUDENT INNER JOIN UNIVERSITY  
ON STUDENT.CITY = UNIVERSITY.CITY;
```

Ключевое слово **INNER** в запросе может быть опущено, так как эта опция в операторе **JOIN** действует по умолчанию.

Рассмотренный выше случай полного соединения (декартова произведения таблиц) с использованием ключевого слова **JOIN** будет выглядеть следующим образом

```
SELECT * FROM STUDENT JOIN UNIVERSITY;
```

что эквивалентно

```
SELECT * FROM STUDENT, UNIVERSITY;
```

Заметим, что в СУБД Oracle задаваемый стандартом языка SQL оператор **JOIN** не поддерживается.