Лабораторная работа №11

**Запросы на соединение отношений. Подзапросы**

*Цель лабораторной работы*: изучение и практическое применение запросов на соединение отношений и подзапросов.

*Оборудование*: компьютерный класс, Microsoft Word, PostgreSQL.

*Теоретическая часть*:

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;

Возможные типы соединений с сопоставлениями строк:

**INNER JOIN**

Для каждой строки R1 из Tаблицы 1 в результирующей таблице содержится строка для каждой строки в Tаблицы 2, удовлетворяющей условию соединения с R1.

**LEFT OUTER JOIN**

Сначала выполняется внутреннее соединение (INNER JOIN). Затем в результат добавляются все строки из Tаблицы 1, которым не соответствуют никакие строки в Таблице2, а вместо значений столбцов Tаблицы 2 вставляются NULL. Таким образом, в результирующей таблице всегда будет минимум одна строка для каждой строки из Tаблицы 1.

**RIGHT OUTER JOIN**

Сначала выполняется внутреннее соединение (INNER JOIN). Затем в результат добавляются все строки из Tаблицы 2, которым не соответствуют никакие строки в Таблице1, а вместо значений столбцов Tаблицы 1 вставляются NULL. Это соединение является обратным к левому (LEFT JOIN): в результирующей таблице всегда будет минимум одна строка для каждой строки из Tаблицы 2.

**FULL OUTER JOIN**

Сначала выполняется внутреннее соединение. Затем в результат добавляются все строки из T1, которым не соответствуют никакие строки в T2, а вместо значений столбцов T2 вставляются NULL. И наконец, в результат включаются все строки из T2, которым не соответствуют никакие строки в T1, а вместо значений столбцов T1 вставляются NULL.

Соединение, использующее предикаты, основанные на равенствах, называется *эквисоединением*. Рассмотренный пример соединения таблиц относятся 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, UNIVERS ITY.UNIV\_NAME, STUDENT.CITY

**FROM** STUDENT **INNERJOIN** 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;

Информация в таблицах STUDENT и EXAM \_ MARKS уже связана посредством поля STUDENT \_ ID. В таблице STUDENT поле STUDENT \_ ID является первичным ключом, а в таблице EXAM \_ MARKS, ссылающимся на него внешним ключом. Состояние связанных таким образом таблиц называется состоянием ссылочной целостности. В данном случае ссылочная целостность этих таблиц подразумевает, что *каждому* значению поля STUDENT \_ ID в таблице EXAM \_ MARKS *обязательно* соответствует *такое же значение* поля STUDENT \_ ID в таблице STUDENT. Другими словами, в таблице EXAM \_ MARKS не может быть записей, имеющих идентификаторы студентов, которых нет в таблице STUDENT. Стандартное применение операции соединения состоит в извлечении данных в терминах этой связи.

Чтобы получить список фамилий студентов с полученными ими оценками и идентификаторами предметов можно использовать следующий запрос:

**SELECT** SURNAME, MARK,SUBJ\_ID

**FROM** STUDENT, EXAM\_MARKS

**WHERE** STUDENT.STUDENT\_ID = EXAM\_MARKS.STUDENT\_ID;

Тот же самый результат может быть получен при использовании в запросе для задания операции соединения таблиц ключевого слова **JOIN**. Запрос с оператором **JOIN**выглядитследующимобразом

**SELECT** SURNAME,MARK

**FROM** STUDENT **JOIN** EXAM\_MARKS

ON STUDENT. STUDENT\_ID = EXAM\_MARKS.STUDENT\_ID;

Хотя выше речь шла о соединении двух таблиц, можно сформировать запросы путем соединения более чем двух таблиц.

Пусть требуется найти фамилии всех студентов, получивших неудовлетворительную оценку, вместе с названиями предметов обучения, по которым получена эта оценка.

**SELECT** SUB J\_NAME, SURNAME, MARK

**FROM** STUDENT, SUBJECT, EXAM\_MARKS

**WHERE** STUDENT.STUDENT\_ID= EXAM\_MARKS.STUDENT\_ID

**AND** SUBJECT. SUB J ID = EXAM MARKS. SUB J ID

**AND** EXAM \_ MARKS . MARK = 2;

To же самое с использованием опера- тора **JOIN**

**SELECT** SUBJ\_NAME, SURNAME, MARK

**FROM** STUDENT **JOIN** SUBJECT

**JOIN** EXAM\_MARKS ON STUDENT.STUDENT\_ID = EXAM\_MARKS.STUDENT\_ID

**AND** SUB JECT.SUB J\_ID = EXAM\_MARKS.SUB J\_ID **AND** EXAM\_MARKS.MARK= 2;

Как отмечалось ранее, при использовании *внутреннего* (**INNER**) соединения таблиц соединяются только те их строки, в которых совпадают значения полей, задаваемые в предложении **WHERE** запроса. Однако во многих случаях это может привести к нежелательной потере информации. Рассмотрим еще раз приведенный выше пример запроса на выборку списка фамилий студентов с полученными ими оценками и идентификаторами предметов. При использовании, как это было сделано в рассматриваемом примере, внутреннего соединения в результат запроса не попадут студенты, которые еще не сдавали экзамены и которые, следовательно, отсутствуют в таблице EXAM \_ MARKS. Если же необходимо иметь записи об этих студентах в выдаваемом запросом списке, то можно присоединить сведения о студентах, не сдававших экзамен, путем использования оператора **UNION** с соответствующим запросом. Например, следующим образом:

**SELECT** SURNAME,**CAST** MARK **AS**CHAR(1), **CAST** SUBJ\_ID **AS** CHAR(IO)

**FROM**STUDENT, EXAM\_MARKS

**WHERE** STUDENT.STUDENT\_ID= EXAM\_MARKS.STUDENT\_ID

**UNIONSELECT** SURNAME,**CASTNULLAS**CHAR(l), **CASTNULLAS** CHAR(IO)

**FROM** STUDENT **WHERE** NOT EXIST

(**SELECT** \* **FROM**EXAM\_MARKS **WHERE**STUDENT.STUDENT\_ID = EXAM\_MARKS.STUDENT\_ID) ;

(здесь функция преобразования типов **CAST** используется для обеспечения совместимости типов полей объединяемых запросов).

Нужный результат, однако, может быть получен и путем использования *внешнего соединения*, точнее одной из его разновидностей – *левого внешнего соединения*, с использованием которого запрос будет выглядеть следующим образом:

**SELECT** SURNAME,MARK**FROM** STUDENT **LEFTOUTER**

**JOIN** EXAM\_MARKS ON STUDENT.STUDENT\_ID = EXAM\_MARKS.STUDENT\_ID;

При использовании *левого* соединения расширение выводимой таблицы осуществляется за счет записей входной таблицы, имя которой указано *слева* от оператора **JOIN**.

Приведенный выше запрос может быть реализован и с применением *правого внешнего соединения*. Он будет иметь следующий вид

**SELECT** SURNAME,MARK

**FROM** EXAM\_MARKS

**RIGHT OUTER JOIN** STUDENT ON EXAM\_MARKS.STUDENT\_ID = STUDENT.STUDENT\_ID;

Здесь таблица STUDENT, за счет записей которой осуществляется расширение выводимой таблицы, стоит справа от оператора **JOIN**

Видно, что использование внешнего правого или левого соединения позволяет существенно упростить запрос, сделать его запись более компактной.

Иногда возникает необходимость включения в результат запроса записей из обеих (правой и левой) соединяемых таблиц, для которых не удовлетворяется условие соединения. Такое соединение называется *полным внешним соединением* и осуществляется указанием в запросе ключевых слов FULL OUTER JOIN или UNION JOIN.

Часто при получении информации из таблиц базы данных необходимо осуществлять соединение таблицы с ее же копией. Например, это требуется в случае, когда требуется найти фамилии студентов, имеющих одинаковые имена. При соединении таблицы с ее же копией вводят псевдонимы (алиасы) таблицы. Запрос для поиска фамилий студентов, имеющих одинаковые имена, выглядит следующим образом

**SELECT** FIRS Т . SURNAME, SECOND. SURNAME

**FROM** STUDENT FIRST, STUDENT SECOND

**WHERE**FIRST . NAME = SECOND . NAME

В этом запросе введены два псевдонима для одной таблицы STUDENT, что позволяет корректно задать выражение, связывающее две копии таблицы. Чтобы исключить повторения строк в выводимом результате запроса из-за повторного сравнения одной и той же пары студентов, необходимо задать порядок следования для двух значений так, чтобы одно значение было меньше, чем другое, что делает предикат асимметричным.

**SELECT** FIRS Т . SURNAME, SECOND. SURNAME

**FROM** STUDENT FIRST, STUDENT SECOND

**WHERE** FIRST.NAME= SECOND.NAME

**AND** FIRST.SURNAME< SECOND.SURNAME.

***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 ');

Рассмотрим 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** CI TY = ' Воронеж ');

Подзапросы можно применять внутри предложения **HAVING**. Пусть требуется определить количество предметов обучения с оценкой, превышающей среднее значение оценки студента с идентификатором 301:

**SELECTCOUNT** (**DISTINCT** SUB JID) , MARK

**FROM** EXAM\_MARKS **GROUP BY** MARK

**HAVING** MARK >( **SELECT AVG** (**MARK**)

**FROM** EXAM\_MARKS **WHERE** STUDENT \_ ID = **301**);

**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 января1999 г.

**SELECT** \* **FROM** SUBJECT SU **WHERE'20/01/1999'**

**IN** (**SELECT** EXAM\_DATE **FROM** EXAM\_MARKS EX

**WHERE** SU.SUB J\_ID = EX.SUB J\_ID);

В некоторых СУБД для выполнения этого запроса, возможно, потребуется преобразование значения даты в символьный тип. В приведенном запросе SU и ЕХ 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 EXAMJMARKS.

Эту же задачу можно решить с помощью операции соединения таблиц:

**SELECTDISTINCT** SU.SUB J\_ID,SUBJ\_NAME, HOUR, SEMESTER

**FROM** SUBJECT FIRST, EXAM\_MARKS SECOND

**WHERE** FIRST.SUBJ\_ID= SECOND.SUB J\_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.

**SELECTDISTINCT** STUDENT\_ID,SURNAME, STIPEND

**FROM** STUDENT El

**WHERE** STIPEND > (**SELEC**T AVG(STIPEND)

**FROM** STUDENT E2 **WHERE**El . KURS = E 2. KURS );

Тот же результат можно получить с помощью следующего запроса:

**SELECTDISTINCT** STUDENT\_ID, SURNAME, STIPEND

**FROM** STUDENT El, (**SELECT** KURS, AVG(STIPEND)

**AS** AVG\_STIPEND **FROM** STUDENT E2

**GROUP BY** E2.KURS) E3

**WHERE** E1.STIPEND>AVG\_STIPENDANDE1.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 Е2) обрабатывается агрегирующей функцией один раз, в результате чего формируется вспомогательная таблица (в запросе она имеет алиас ЕЗ), со строками которой затем соединяются строки первой таблицы (алиас Е1). Следует иметь в виду, что реальное время выполнения запроса в большой степени зависит от оптимизатора запросов конкретной СУБД.

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 В

**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.

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 В

**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** SUBJID

**FROM** EXAM\_MARKS A **WHEREEXISTS** (**SELECT** \*

**FROM** EXAM\_MARKS В**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 S E COND **GROUP BY** SUBJID

**HAVING**COUNT(SUBJID) **> 1 WHERE** FIRST.STUDENT\_ID= SECOND.STUDENT\_ID);

Операторы сравнения с множеством значений имеют следующий смысл. IN *Равно* любому из значений, полученных во внутреннем запросе. NOT IN *He равно* ни одному из значений, полученных во внутреннем запросе.

= ANY To же, что и IN . Соответствует логическому оператору OR .

> ANY , > = ANY

*Больше, чем* (либо *больше или равно*) любое полученное число. Эквивалент- но > или > = для самого меньшего полученного числа.

< ANY , < = ANY

*Меньше, чем* (либо *меньше или равно*) любое полученное число. Эквивалент < или < = для самого большего полученного числа.

= ALL

Равно всем полученным значениям. Эквивалентно логическому оператору AND .

> ALL , > = ALL

*Больше, чем* (либо *больше или равно*) все полученные числа. Эквивалент > или > = для самого большего полученного числа.

< ALL , < = ALL

*Меньше, чем* (либо *меньше или равно*) все полученные числа. Эквивалентно < или < =самого меньшего полученного числа.

Следует иметь в виду, что в некоторых СУБД поддерживаются не все из этих операторов.

Примеры запросов с использованием приведенных операторов.

Выбрать сведения о студентах, проживающих в городе, где расположен университет, в котором они учатся.

**SELECT** \* **FROM** STUDENT S

**WHERE** CITY **= ANY** (**SELECT** CITY **FROM** UNIVERSITY U

**WHERE** U.UNIV\_ID = S.UNIV\_ID);

Другой вариант этого запроса

**SELECT** \* **FROM** STUDENT S

**WHERE** CITYIN (**SELECT** CITY **FROM** UNIVERSITY U

**WHERE** U.UNIV\_ID = S.UNIV\_ID);

Выборка данных об идентификаторах студентов, у которых оценки превосходят величину, по крайней мере, одной из оценок, полученных ими же 6 октября 1999 года.

**SELECT DISTINCT** STUDENT\_ID

**FROM** EXAM\_MARKS**WHERE** MARK **> ANY** (**SELECT** MARK

**FROM** EXAM\_MARKS **WHERE** EXAMDATE = '06/10/1999');

Оператор **ALL**, как правило, эффективно используется с неравенствами, а не с равенствами, поскольку значение *равно всем*, которое должно получиться в этом случае в результате выполнения подзапроса, может иметь место, только если все результаты идентичны. Такая ситуация практически не может быть реализована, так как, если подзапрос генерирует множество различных значений, то никакое одно значение не может быть равно сразу всем значениям в обычном смысле. В SQL выражение **<> ALL** реально означает *не равно ни одному* из результатов подзапроса.

Подзапрос, выбирающий данные о названиях всех университетов с рейтингом более высоким, чем рейтинг любого университета в Воронеже:

**SELECT** \* **FROM** UNIVERSITY

**WHERE** RATING **> ALL** (**SELECT** RATING

**FROM** UNIVERSITY **WHERE** CI TY= ' Воронеж ');

В этом запросе вместо **ALL** можно также использовать **ANY**. (Проанализируйте, как в этом случае изменится смысл приведенного запроса?)

**SELECT** \* **FROM** UNIVERSITY

**WHERE** NOTRATING**>ANY** (**SELECT** RATING

**FROM** UNIVERSITY **WHERE**CITY = ' Воронеж ');

Необходимо иметь в виду, что при обработке NULL -значений следует учитывать различие реакции на них операторов **EXISTS**, **ANY** и **ALL** .

Когда правильный подзапрос не генерирует никаких выходных данных, оператор **ALL** автоматически принимает значение **истина**, а оператор **ANY** - значение **ложь**.

Запрос

**SELECT** \* **FROM** UNIVERSITY

**WHERE** RATING **> ANY** ( **SELECT** RATING

**FROM** UNIVERSITY **WHERE** CITY = ' New York');

не генерирует выходных данных (подразумевается, что в базе нет данных об университетах из города NewYork ), в то время как запрос

**SELECT** \* **FROM** UNIVERSITY

**WHERE** RATING **> ALL** (**SELECT** RATING

**FROM** UNIVERSITY **WHERE** CITY = ' New York');

полностью воспроизведет таблицу UNIVERSITY.

Использование NULL-значений создает определенные проблемы для рассматриваемых операторов. Когда в SQL сравниваются два значения, одно из которых NULL -значение, результат принимает значение UNKNOWN (неизвестно). Предикат UNKNOWN, так же как и FALSE -предикат, создает ситуацию, когда строка не включается в состав выходных данных, но результат при этом будет различен для разных типов запросов, в зависимости от использования в них ALL или ANY вместо EXISTS. Рассмотрим в качестве примера две реализации запроса: найти все данные об университетах, рейтинг которых меньше рейтинга любого университета в Москве.

1. **SELECT \* FROM** UNIVERSITY

**WHERE** RATING < ANY (**SELECT** RATING **FROM** UNIVERSITY

**WHERE** CITY= ' Москва ');

1. **SELECT \* FROM** UNIVERSITYA

**WHERE NOT EXISTS** (**SELECT \* FROM** UNIVERSITY В

**WHERE** A. RATING >= В.RATING **AND** B.CITY=' Москва ');

При отсутствии в таблицах NULL оба эти запроса ведут себя совершенно одинаково. Пусть теперь в таблице UNIVERSITY есть строка с NULL - значениями в столбце RATING. В версии запроса с ANY в основном запросе, когда выбирается поле RATING с NULL, предикат принимает значение UNKNOWN и строка не включается в состав выходных данных. Во втором же варианте запроса, когда NOT EXISTS выбирает эту строку в основном запросе, NULL -значение используется в предикате подзапроса, присваивая ему значение UNKNOWN. Поэтому в результате выполнения подзапроса не будет получено ни одного значения и подзапрос примет значение ложь. Это в свою очередь сделает NOT EXISTS истинным, и, следовательно, строка с NULL значением в поле RATING попадет в выходные данные. По смыслу запроса такой результат является неправильным, так как на самом деле рейтинг университета, описываемого данной строкой может быть и больше рейтинга какого-либо московского университета (он просто неизвестен). Указанная проблема связана с тем, что значение EXISTS всегда принимает значения истина или ложь, и никогда – UNKNOWN. Это является доводом для использования в таких случаях оператора ANY вместо EXISTS.

При отсутствии NULL-значений оператор EXISTS может быть использован вместо ANY и ALL. Также вместо EXISTS и NOT EXISTS могут быть использованы те же самые подзапросы, но с использованием COUNT (\*) в предложении SELECT. Например, запрос

**SELECT** \* **FROM** UNIVERSITY A

**WHERENOT EXISTS** (**SELECT**\* **FROM** UNIVERSITY В

**WHERE** A.RATING > = В.RAT ING AND B .CITY = ' Москва ');

может быть представлен и в следующем виде

**SELECT** \* FROM UNIVERSITY A

**WHERE** 1 > (**SELECT**COUNT(\*) **FROM** UNIVERSITY В

**WHERE** A.RATING > = В.RATING **AND** B.CITY= ' Москва ');

Более подробная информация:

https://postgrespro.ru/docs/postgresql/17/index или

https://postgrespro.ru/docs/postgrespro/17/index

***Практическая часть*:**

***Указания к выполнению заданий:***

***1. Работа производится в созданных в предыдущих работах таблицах.***

***2. Сначала необходимо формулировать смысл запроса на языке, понятном пользователю в данной предметной области, например, «Запрос, выбирающий данные о фамилии, имени и номере курса для студентов, получающих стипендию больше 1400», потом привести сам запрос, а затем результат выполнения запроса в виде скриншота.***

***3. Данные (и параметры в условии выборки) должны быть подобраны таким образом, чтобы в результирующей выборке было 3-4 записи.***

1. Напишите запрос, демонстрирующий соединение двух таблиц с помощью конструкции SELECT ... FROM TABLE1, TABLE2 WHERE... . Перепишите тот же запрос с помощью конструкции JOIN. Убедитесь, что результаты выполнения запросов одинаковы.
2. Напишите запрос, демонстрирующий смысл и назначение конструкции LEFT JOIN. Перепишите его с помощью конструкции RIGHT JOIN. Убедитесь, что результаты выполнения запросов одинаковы.
3. Напишите запрос, в котором таблица соединяется (JOIN) сама с собой.
4. Напишите запрос, в котором агрегация происходит по результату соединения таблиц. То есть, в запросе должны присутствовать агрегирующая функция (SUM, AVG, MAX, MIN или COUNT), GRUOP BY и HAVING, WHERE и JOIN (внутренний или внешний).
5. Напишите пример запроса, в котором вместо любой из констант выражения, определяющего условие WHERE, используется скалярный подзапрос.
6. Напишите запрос с векторным подзапросом (ключевое слово IN)
7. Напишите два запроса с ключевыми словами ANY и ALL, делающие одно и то же.
8. Напишите пример запроса с табличным подзапросом (ключевое слово EXISTS).
9. Напишите пример запроса несколькими уровнями вложенности.
10. Напишите пример запроса, в котором вместо таблицы, указываемой после ключевого слова FROM, используется подзапрос.
11. Напишите пример связанного подзапроса.