- 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');
```

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

```
SELECT COONT(DISTINCT SUBJ_IDj, MARK
   FROM EXAM_MARKS
   GROUP BY MARK
   HAVING MARK >
```

```
(SELECT AVG(MARK)
FROM EXAM_MARKS
WHERE STUDENT_ID = 301);
```

2.9. 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
      PROM EXAM_MARKS EX
      WHERE SU.SUBJ_ID = EX.SUBJ_ID);
```

В некоторых СУБД для выполнения этого запроса может потребоваться 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, SUBJJAME, HOUR, SEMESTER
   PROM 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 (STIPENDj 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.

2.10. Связанные подзапросы в HAVING

В разделе 2.4 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.

Упражнения

1. Напишите **запрос с подзапросом для** получения **данных обо всех** оценках студента **с** фамилией «Иванов». Предположим, что его персональный **номер** неизвестен. **Всегда** ли такой **запрос** будет корректным?
2. Напишите запрос, выбирающий **данные об** именах **всех** студентов, имеющих **по** предмету **с** идентификатором **101** балл **выше общего среднего** балла.
3. Напишите запрос, который выполняет выборку имен **всех** студентов, имеющих **по** предмету **с** идентификатором **102** балл ниже **общего среднего** балла
4. Напишите запрос, выполняющий **вывод** количества предметов, по **которым** экзаменовался каждый студент, сдававший более **20** предметов.
5. Напишите команду SELECT, использующую **связанные** подзапросы и выполняющую **вывод** имен **и** идентификаторов студентов,

у которых стипендия совпадает с максимальным значением стипендии для города, в котором живет студент.

6. Напишите запрос, который позволяет вывести имена и идентификаторы всех студентов, для которых точно известно, что они проживают в городе, где нет ни одного университета.
7. Напишите два запроса, которые позволяют вывести имена и идентификаторы всех студентов, для которых точно известно, что они проживают не в том городе, где расположен их университет. Один запрос с использованием соединения, а другой — с использованием связанного подзапроса.

2.11. 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 (не известно).

Пусть, например, нужно извлечь из таблицы EXAM_MARK 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_IDj > 1
       WHERE FIRST.STUDENT ID = SECOND.STUDENT ID);
```

Упражнения

1. Напишите запрос с EXISTS, позволяющий вывести данные обо всех студентах, обучающихся в вузах, которые имеют рейтинг выше 300
2. Напишите предыдущий запрос, используя соединения.
3. Напишите запрос с EXISTS, выбирающий сведения обо всех студентах, для которых в том же городе, где живет студент, существуют университеты, в которых он не учится.
4. Напишите запрос, выбирающий из таблицы SUBJECT данные о названиях предметов обучения, экзамены по которым *сданы* более чем одним студентом.

2.12. Операторы сравнения с множеством значений IN, ANY, All

Операторы сравнения с множеством значений имеют следующий смысл.

IN	*Равно* любому из значений, полученных во внутреннем запросе.
NOT IN	*Не равно* ни одному из значений, полученных во внутреннем запросе.
= ANY	То же, что и IN, Соответствует логическому оператору OR.
> ANY, > = ANY	*Больше, чем* (либо *больше или равно)* любое полученное число. Эквивалентно > 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 CITY IN
       (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 EXAM DATE = '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 Воронежа:

```
SELECT *
   FROM UNIVERSITY
   WHERE RATING > ALL
       (SELECT RATING
       FROM UNIVERSITY
       WHERE CITY = 'Воронеж') ;
```

В этом запросе вместо ALL можно использовать ANY (проанализируйте, как в этом случае изменится смысл приведенного запроса):

```
SELECT *
   FROM UNIVERSITY
   WHERE NOT RATING > ANY
       (SELECT RATING
       FROM UNIVERSITY
       WHERE CITY = 'Воронеж');
```

2.13. Особенности применения операторов ANY, ALL, EXISTS при обработке пустых значений (NULL)

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 города New York), в то время как запрос

```
SKLECT *
   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');
2) SELECT *
     FROM UNIVERSITY A
     WHERE NOT EXISTS
        (SELECT *
        FROM UNIVERSITY B
        WHERE A.RATING >= B.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.

2.14. Использование COUNT 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
   WHERE NOT EXISTS
       (SELECT *
       FROM UNIVERSITY B
       WHERE A.RATING > = B.RATING
       AND B.CITY = 'Москва');
```

может быть представлен и в следующем виде:

```
SELECT *
   FROM UNIVERSITY A
   WHERE 1 >
       (SELECT COUNT(*)
       FROM UNIVERSITY B
       WHERE A.RATING > = B.RATING
       AND B.CITY = 'Москва');
```

Упражнения

1. Напишите запрос, выбирающий данные о названиях университетов, рейтинг которых равен или превосходит рейтинг Воронежского государственного университета.
2. Напишите запрос, использующий ANY или ALL, выполняющий выборку данных о студентах, у которых в городе их постоянного местожительства нет университета.
3. Напишите запрос, выбирающий из таблицы EXAM_MARKS данные о названиях предметов обучения, для которых значение полученных на экзамене оценок (поле MARK) превышает любое значение оценки для предмета, имеющего идентификатор, равный 105.
4. Напишите этот же запрос с использованием MAX.

2.15. 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, ST0DENT_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 так же, 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.

2.16. Устранение дублирования в UNION

В отличие от обычных запросов UNION автоматически исключает из выходных данных дубликаты строк, например, в запросе

```
SELECT CITY
   FROM STUDENT
UNION
SELECT CITY
   FROM LECTURER;
```

совпадающие наименования городов будут исключены.

Если все же необходимо в каждом запросе вывести все строки независимо от того, имеются ли *такие* же строки в других объединяемых запросах, то следует использовать во множественном запросе конструкцию с оператором UNION ALL. Так, в запросе

```
SELECT CITY
   FROM STUDENT
UNION ALL
SELECT CITY
   FROM LECTURER;
```

дубликаты значений городов, выводимые второй частью запроса, не будут исключаться.

Приведем еще один пример использования оператора UNION. Пусть необходимо составить отчет, содержащий для каждой даты сдачи экзаменов сведения по каждому студенту, получившему максимальную или минимальную оценки.

```
SELECT 'МЭКСОЦ', A.STUDENT_ID, SURNAME, MARK, EXAM_DATE
   FROM STUDENT A, EXAM_MARKS B
   WHERE  (A.STUDENT_ID = B.STUDENT_ID
      AND B.MARK =
          (SELECT MAX(MARK)
          FROM EXAM_MARKS C
          WHERE C.EXAM_DATE = B.EXAM_DATE))
UNION ALL
SELECT 'МИНОЦ', A.STUDENT_ID, SURNAME, MARK, EXAM_DATE
   FROM STUDENT A, EXAM_MARKS B
   WHERE  (A.STUDENT_ID = B.STUDENT_ID           \
      AND B.MARK =
          (SELECT MIN(MARK)
          FROM EXAM_MARKS C
          WHERE C.EXAM_DATE = B.EXAM_DATE));
```

Для отличия строк, выводимых первой и второй частями запроса, в них вставлены текстовые константы 'Макс оц' и 'Мин оц'.

В приведенном запросе агрегирующие функции используются в подзапросах. Это является нерациональным с точки зрения времени, затрачиваемого на выполнение запроса (см. раздел 2.9). Более эффективна форма запроса, возвращающего аналогичный результат:

```
SELECT 'МаКСОЦ', A.STUDENT_ID, SURNAME, E.MARK, E.EXAM_DATE
   FROM STUDENT A,
       (SELECT B.STUDENT_ID, B.MARK, B.EXAM_DATE
          FROM EXAM_MARKS B,
              (SELECT MAX(MARK) AS MAX_MARK, C.EXAM_DATE
                 FROM EXAM__MARKS C
             GROUP BY C.EXAM_DATE) D
          WHERE B.EXAM_DATE=D.EXAM_DATE
                 AND B.MARK=MAX_MARK) E
   WHERE A.STUDENT_ID=E.STUDENT_ID
UNION ALL
SELECT 'МИНОЦ', A. STUDENT__ID, SURNAME, E .MARK, E .EXAM_DATE
   FROM STUDENT A,
       (SELECT  B.STUDENT_ID, B.MARK, B.EXAM_DATE
          FROM EXAM_MARKS B,
              (SELECT MIN(MARK) AS MIN_MARK, C.EXAM_DATE
                 FROM EXAM_MARKS C
             GROUP BY C.EXAM_DATE) D
          WHERE B.EXAM_DATE=D.EXAM_DATE
      AND B.MARK=MIN_MARK) E
   WHERE A.STUDENT_ID=E.STUDENT_ID
```

2.17. Использование UNION с ORDER BY

Предложение ORDER BY применяется для упорядочения выходных данных объединения запросов так же, как и для отдельных запросов. Последний пример, при необходимости упорядо-

чения выходных данных запроса по фамилиям студентов и датам экзаменов, может выглядеть следующим образом:

```
SELECT 'МЭКСОЦ', A.STUDENT_ID, SURNAME, E.MARK, E.EXAM_DATE
   FROM STUDENT A,
       (SELECT B.STUDENT_ID, B.MARK, B.EXAM_DATE
          FROM EXAM_MARKS B,
              (SELECT MAX (MARK) AS MAX_MARK, C.EXAM_DATE
                 FROM EXAM_MARKS C
                 GROUP BY C.EXAM_DATE) D
          WHERE B.EXAM_DATE=D.EXAM_DATE
                 AND B.MARK=MAX_MARK) E
   WHERE A.STUDENT_ID=E.STUDENT_ID
UNION ALL
SELECT 'МИНОЦ', A.STUDENT_ID, SURNAME, E.MARK, E.EXAM_DATE
   FROM STUDENT A,
       (SELECT B.STUDENT_ID, B.MARK, B.EXAM_DATE
          FROM EXAM_MARKS B,
              (SELECT MIN(MARK) AS MIN_MARK, C.EXAM_DATE
              FROM EXAM_MARKS C
              GROUP BY C.EXAM_DATE) D
          WHERE B.EXAM_DATE=D.EXAM_DATE
      AND B.MARK=MIN_MARK) E
WHERE A.STUDENT_ID=E.STUDENT_ID
ORDER BY SURNAME,E.EXAM_DATE;
```

2.18. Внешнее объединение

Часто бывает 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 наименова-

Соединение, использующее предикаты, основанные на равенствах, называется *эквисоединением*. Рассмотренный пример соединения таблиц относятся к виду так называемого *внутреннего* (INNER) *соединения*. При этом соединяются только те строки таблиц, для которых истинным является 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.

2.19.1. Операции соединения таблиц посредством ссылочной целостности

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 результат может быть получен при использовании в запросе для задания операции соединения таблиц ключевого слова JOIN. Запрос с оператором JOIN выглядит следующим образом:

```
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 SUBJ_NAME, SURNAME, MARK
   FROM STUDENT,SUBJECT,EXAM_MARKS
   WHERE STUDENT.STUDENT_ID = EXAM_MARKS.STUDENT_ID
      AND  SUBJECT.SUBJ_ID  =  EXAM_MARKS.SUBJ_ID
      AND  EXAM_MARKS.MARK  =  2;
```

То же самое с использованием оператора JOIN:

```
SELECT SUBJ_NAME, SURNAME, MARK
   FROM STUDENT JOIN SUBJECT JOIN EXAM_MARKS
   ON  STUDENT.STUDENT_ID  =  EXAM_MARKS.STUDENT_ID
      AND  SUBJECT.SUBJ_ID  =  EXAM_MARKS.SUBJ_ID
      AND  EXAM MARKS.MARK  =  2;
```

2.19.2. Внешнее соединение таблиц

Как отмечалось ранее, при использовании *внутреннего* (INNER) соединения таблиц соединяются только те их строки, в которых совпадают значения полей, задаваемые в запросе предложением WHERE. 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 EXAMJMARKS. 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 CHARdV  CAST SUBJ_ID AS CHAR (10)
   FROM STUDENT,EXAM_MARKS
   WHERE STUDENT.STUDENT_ID = EXAM_MARKS.STUDENT_ID
UNION
SELECT SURNAME, CAST NULL AS CHAR(1),  CAST NULL AS CHAR(10)
   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 может быть получен и путем использования *внешнего соединения,* точнее, одной из его разновидностей — *левого внешнего соединения,* с применением которого запрос будет выглядеть следующим образом:

```
SELECT SURNAME, MARK
   FROM STUDENT LEFT OUTER 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.

Следует заметить, что нотация запросов с внешним соединением в СУБД Oracle отличается от приведенной нотации, задаваемой стандартом языка SQL. В нотации, используемой в Oracle, этот же запрос будет иметь вид:

```
SELECT SURNAME, MARK, SUBJ_ID
   FROM STUDENT,EXAM_MARKS
   WHERE STUDENT.STUDENT_ID = EXAM_MARKS.STUDENT_ID(+);
```

Знак (+) ставится у той таблицы, которая дополняется записями с NULL-значениями, чтобы при соединении таблиц в выходное отношение попали и те записи другой таблицы, для которых в таблице со знаком (+) не находится строк с соответствующими значениями атрибутов, используемых для соединения. То есть для *левого* внешнего соединения (по нотации стандарта SQL) в запросе Oracle-SQL указатель (+) ставится у *правой* таблицы.

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, указана справа от оператора JOIN.

В нотации Oracle этот запрос будет выглядеть следующим образом:

```
SELECT SURNAME, MARK, SUBJ_ID
   FROM STUDENT,EXAM_MARKS
   WHERE EXAM_MARKS.STUDENT_ID(+) = STUDENT.STUDENT_ID;
```

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.

Упражнения

1. Напишите запрос, который выполняет вывод данных о фамилиях *сдававших* экзамены студентов (вместе с идентификаторами каждого сданного ими предмета обучения).
2. Напишите запрос, который выполняет выборку значений фамилии *всех* студентов с указанием для студентов, сдававших экзамены, идентификаторов сданных ими предметов обучения.
3. Напишите запрос, который выполняет вывод данных о фамилиях студентов, *сдававших* экзамены, вместе с наименованиями каждого сданного ими предмета обучения.
4. Напишите запрос на выдачу для каждого студента названий всех предметов обучения, по которым этот студент получил оценку 4 или 5.
5. Напишите запрос на выдачу данных о названиях всех предметов, по которым студенты получили только хорошие (4 и 5) оценки. В выходных данных должны быть приведены фамилии студентов, названия предметов и оценка.
6. Напишите запрос, который выполняет вывод списка университетов с рейтингом, превышающим 300, вместе со значением максимального размера стипендии, получаемой студентами в этих университетах.
7. Напишите запрос на выдачу списка фамилий студентов (в алфавитном порядке) вместе со значением рейтинга университета, где каждый из них учится, включив в список и тех студентов, для которых в базе данных не указано место их учебы.

2.19.3. Использование псевдонимов при соединении таблиц

Часто при запросе информации необходимо осуществлять соединение таблицы с ее же копией. Например, это требуется в случае, когда нужно найти фамилии студентов, имеющих оди-

наковые имена. При соединении таблицы с ее же копией вводят псевдонимы (алиасы) таблицы. Запрос для поиска фамилий студентов, имеющих одинаковые имена, выглядит следующим образом:

```
SELECT FIRST.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 FIRST. SURNAME, SECOND. SURNAME
<j
f1      FROM STUDENT FIRST, STUDENT SECOND
Ч
;!     WHERE FIRST.NAME = SECOND.NAME
          AND  FIRST.SURNAME  <  SECOND.SURNAME
```

Упражнения

1. Написать запрос, выполняющий вывод списка всех пар фамилий студентов, проживающих в одном городе. При этом не включать в список комбинации фамилий студентов самих с собой (то есть комбинацию типа «Иванов-Иванов») и комбинации фамилий студентов, отличающиеся порядком следования (то есть включать одну из двух комбинаций типа «Иванов-Петров» и «Петров-Иванов»).
2. Написать запрос, выполняющий вывод списка всех пар названий университетов, расположенных в одном городе, не включая в список комбинации названий университетов самих с собой и пары названий университетов, отличающиеся порядком следования.
3. Написать запрос, который позволяет получить данные о названиях университетов и городов, в которых они расположены, с рейтингом, равным или превышающим рейтинг ВГУ.

3 Манипулирование данными

3.1. Команды манипулирования данными

В SQL для выполнения операций ввода данных в таблицу, **их** изменения и удаления предназначены три команды языка манипулирования данными (DML). Это команды INSERT (вставить), UPDATE (обновить), DELETE (удалить).

Команда INSERT осуществляет **вставку** в таблицу новой • строки. **В** простейшем случае она имеет вид:

INSERT INTO *<имя таблицы>* VALUES *(<значение>, <значение>,);*

При такой записи указанные в скобках после ключевого слова VALUES значения вводятся в поля добавленной в таблицу новой строки в том порядке, **в** котором соответствующие столбцы указаны при создании таблицы, то есть **в** операторе CREATE TABLE.

Например, ввод новой строки в таблицу STUDENT может быть осуществлен следующим образом:

```
INSERT INTO STUDENT
VALUES (101,'Иванов','Александр', 200, 3,'Москва',
                                        '6/10/1979', 15);
```

Чтобы такая команда могла быть выполнена, таблица с указанным в ней именем (STUDENT) должна быть предварительно определена (создана) командой CREATE TABLE. Если в какое-либо поле необходимо вставить NULL-значение, то оно вводится как обычное значение:

```
INSERT INTO STUDENT
VALUES (101, 'Иванов', NULL, 200, 3, 'Москва','6/10/1979', 15);
```

В случаях, когда необходимо ввести значения полей в порядке, отличном от порядка столбцов, заданного командой CREATE TABLE, или требуется ввести значения не во все столбцы, следует использовать следующую форму команды INSERT:

```
INSERT INTO STUDENT (STUDENT_ID, CITY, SURNAME, NAME)
VALUES (101, 'Москва', 'Иванов', 'Саша');
```

Столбцам, наименования которых не указаны в приведенном в скобках списке, автоматически присваивается значение по умолчанию, если оно назначено при описании таблицы (команда CREATE TABLE), либо значение NULL.

С помощью команды INSERT можно извлечь значение из одной таблицы и разместить его в другой, например, запросом следующего вида:

```
INSERT INTO STUDENT1
   SELECT *
      FROM STUDENT
      WHERE CITY = 'Москва';
```

При этом таблица STUDENTI должна быть предварительно создана командой CREATE TABLE (раздел 4.1) и иметь структуру, идентичную таблице STUDENT.

Удаление строк из таблицы осуществляется с помощью команды DELETE.

Следующее выражение удаляет все строки таблицы EXAM_MARKSI .

```
DELETE FROM EXAM_MARKS1;
```

В результате таблица становится пустой (после этого она может быть удалена командой DROP TABLE).

Для удаления из таблицы сразу нескольких строк, удовлетворяющих некоторому условию, можно воспользоваться предложением WHERE, например:

```
DELETE FROM EXAM_MARKS1
   WHERE STUDENT_ID = 103;
```

Можно удалить группу строк:

```
DELETE FROM STUDENTI
   WHERE CITY = 'Москва';
```

Команда UPDATE позволяет **изменять,** то есть обновлять значения некоторых или всех полей в существующей строке или строках таблицы. Например, чтобы для всех университетов, све-

дения о которых находятся в таблице UNIVERSITYI, изменить рейтинг на значение 200, можно использовать конструкцию:

```
UPDATE UNIVERSITYI
SET RATING = 200;
```

Для указания конкретных строк таблицы, значения полей которых должны быть изменены, в команде UPDATE можно использовать предикат, указываемый в предложении WHERE.

```
UPDATE UNIVERSITYI
SET RATING = 200
WHERE CITY = 'Москва';
```

В результате выполнения этого запроса будет изменен рейтинг только у университетов, расположенных в Москве.

Команда UPDATE позволяет изменять не только один, но и множество столбцов. Для указания конкретных столбцов, значения которых должны быть модифицированы, используется предложение SET.

Например, наименование предмета обучения 'Математика' (для него SUBJ_ID = 43) должно быть заменено на название **'Высшая** математика', при этом идентификационный номер необходимо сохранить, но в соответствующие поля строки таблицы ввести новые данные об этом предмете обучения. Запрос будет выглядеть следующим образом:

```
UPDATE SUBJECT1
SET SUBJ_NAME = Высшая математика', HOUR = 36, SEMESTER = 1
WHERE SUBJ_ID = 43;
```

В предложении SET команды UPDATE можно использовать скалярные выражения, указывающие способ изменения значений поля, в которые могут входить значения изменяемого и других полей.

```
UPDATE UNIVERSITYI
SET RATING = RATING*2;
```

Например, для увеличения в таблице STUDENTi значения поля STIPEND в два раза для студентов из Москвы можно использовать запрос

```
UPDATE STUDENT1
SET STIPEND = STIPEND*2
WHERE CITY = 'Москва';
```

Предложение SET не является предикатом, поэтому в нем можно указать значение NULL следующим образом:

```
UPDATE UNIVERSITY1
SET RATING = NULL
WHERE CITY = 'Москва';
```

Упражнения

1. Напишите команду, которая вводит в таблицу SUBJECT строку для нового предмета обучения со следующими значениями полей:
   SEMESTER = 4; SUBJ_NAME = **'Алгебра';** HOUR = 72; SUBJ_ID =201.
2. Введите запись для нового студента, которого зовут Орлов Николай, обучающегося на первом курсе ВГУ, живущего в Воронеже, сведения о дате рождения и размере стипендии неизвестны.
3. Напишите команду, удаляющую из таблицы EXAM_MARKS записи обо всех оценках студента, идентификатор которого равен 100.
4. Напишите команду, которая увеличивает на 5 значение рейтинга всех имеющихся в базе данных университетов, расположенных в Санкт-Петербурге.
5. Измените в таблице значение города, в котором проживает студент Иванов, на «Воронеж».

3.2. Использование подзапросов в INSERT

Применение оператора INSERT с подзапросом позволяет загружать сразу несколько строк в одну таблицу, используя информацию из другой таблицы. В то время как оператор INSERT, использующий VALUES, добавляет только одну строку, INSERT с подзапросом добавляет в таблицу столько строк, сколько подзапрос извлекает из другой таблицы. При этом количество и тип возвращаемых подзапросом столбцов должно соответствовать количеству и типу столбцов таблицы, в которую вставляются данные.

Например, пусть таблица STUDENTI имеет структуру, полностью совпадающую со структурой таблицы STUDENT. Запрос, позволяющий заполнить таблицу STUDENTI записями обо всех студентах из Москвы из таблицы STUDENT, выглядит следующим образом:

```
INSERT INTO STUDENTI
   •SELECT *
      FROM STUDENT
      WHERE CITY = 'Москва';
```

Для того же, чтобы добавить в таблицу STUDENTI сведения обо всех студентах, которые *учатся* в Москве, можно использовать в предложении WHERE соответствующий подзапрос. Например,

```
INSERT INTO STUDENTI
   SELECT *
      FROM STUDENT
      WHERE UNIV_ID IN
       (SELECT UNIV_ID
      FROM UNIVERSITY
      WHERE CITY = 'Москва');
```

3.2.1. Использование подзапросов, основанных на таблицах внешних запросов

Предположим, существует таблица SSTUD, в которой хранятся сведения о студентах, обучающихся в том же городе, в котором они живут. Можно заполнить эту таблицу данными из таблицы STUDENT, используя связанные подзапросы, следующим образом:

```
INSERT INTO SSTUD
SELKCT *
FROM STUDENT A
WHERE CITY IN
 (SELECT CITY
FROM UNIVERSITY B
WHERE A.UNIV ID = B.UNIV ID);
```

Предположим, требуется выбрать список студентов, имеющих максимальный балл на каждый день сдачи экзаменов, и разместить его в другой таблице с именем EXAM. Это можно осуществить с помощью запроса

```
INSERT INTO EXAM
SELECT EXAM_ID, STUDENT_ID, SUBJ_ID, MARK, EXAMJ5ATE
FROM EXAM_MARKS A
WHERE MARK =
 (SELECT MAX(MARK;
FROM EXAM_MARKS B
WHERE A.EXAM_DATE = B.EXAM_DATE);
```

3.2.2. Использование подзапросов с DELETE

Пусть филиал университета в Нью-Васюках ликвидирован и требуется удалить из таблицы STUDENT записи о студентах, которые там учились. Эту операцию можно выполнить с помощью запроса

```
DELETE
FROM STUDENT
WHERE UNIV_ID IN
 (SELECT UNIV_ID
FROMUNIVERSITY
WHERE CITY = 'Нью-Васюки');
```

В предикате предложения FROM (подзапроса) нельзя ссылаться на таблицу, из которой осуществляется удаление. Однако можно ссылаться на текущую строку из таблицы, являющуюся кандидатом на удаление, то есть на строку, которая в настоящее время проверяется в основном предикате.

```
DELETE
FROM STUDENT
WHERE EXISTS
 (SELECT *
FROM UNIVERSITY
```

```
WHERE RATING = 401
AND STUDENT.UNIV_ID = UNIVERSITY.UNIV_ID);
```

Часть AND предиката внутреннего запроса ссылается на таблицу STUDENT. Команда удаляет данные о студентах, которые учатся в университетах, имеющих рейтинг, равный 401. Существуют и другие способы решения этой задачи.

```
DELETE
FROM STUDENT
WHERE 401 IN
 (SELECT RATING
FROM UNIVERSITY
WHERE STUDENT.UNIV_ID = UNIVERSITY.UNIV__ID);
```

Пусть нужно найти наименьшее значение оценки, полученной в каждый день сдачи экзаменов, и удалить из таблицы сведения о студенте, который получил эту оценку. Запрос будет иметь вид:

```
DELETE
FROM STUDENT
WHERE STUDENT_ID IN
 (SELECT STUDENT_ID
FROM EXAM_MARKS A
WHERE MARK=
 (SELECT MIN(MARK)
FROM EXAM_MARKS B
WHERE A.EXAM_DATE = B.EXAMJDATE));
```

Так как столбец STUDENT_ID является первичным ключом, то удаляется единственная строка.

Если в какой-то день сдавался только один экзамен (то есть получена только одна минимальная оценка) и по какой-либо причине запись, в которой находится эта оценка, требуется оставить, то решение будет иметь вид:

```
DELETE
FROM STUDENT
```

```
WHERE STUDENT_ID IN
 (SELECT STUDENT_ID
FROM EXAM_MARKS A
WHERE MARK =
 (SELECT MIN(MARK)
FROM EXAM_MARKS B
WHERE A.EXAM_DATE = B.EXAM_DATE
AND 1 <
 (SELECT COONT(SUBJ_ID)
FROM EXAM_MARKS B
WHERE A.EXAM_DATE = B.EXAM_DATE)));
```

3.2.3. Использование подзапросов с UPDATE

С помощью команды UPDATE можно применять подзапро-- сы в любой форме, приемлемой для команды DELETE.

Например, используя связанные подзапросы, можно увеличить значение размера стипендии на 20 в записях студентов, сдавших экзамены на 4 и 5.

```
UPDATE STUDENT1
SET STIPEND = STIPEND + 20
WHERE 4 <=
 (SELECT MIN(MARK)
FROM EXAM_MARKS
WHERE EXAM_MARKS.STUDENT_ID = STUDENT1.STUDENT_ID);
```

Другой запрос: «Уменьшить величину стипендии на 20 всем студентам, получившим на экзамене минимальную оценку».

```
UPDATE STUDENT1
SET STIPEND = STIPEND - 20
WHERE STUDENT_ID IN
 (SELECT STUDENT_ID
FROM EXAM_MARKS A
WHERE MARK =
```

```
(SELECT MIN(MARK)
FROM EXAM_MARKS B
WHERE A.EXAM DATE = B.EXAM DATE));
```

Упражнения

1. Пусть существует таблица с именем STUDENTI, определения столбцов которой полностью совпадают с определениями столбцов таблицы STUDENT. Вставить в эту таблицу сведения о студентах, успешно сдавших экзамены более чем по пяти предметам обучения.
2. Напишите команду, удаляющую из таблицы SUBJECTI сведения о предметах обучения, по которым студентами не получено ни одной оценки.
3. Напишите запрос, увеличивающий данные о величине стипендии на 20% всем студентам, у которых общая сумма баллов превышает значение 50.

4 Создание объектов базы данных

4.1. Создание таблиц базы данных

Создание объектов базы данных осуществляется с помощью операторов языка определения данных (DDL).

Таблицы базы данных создаются с помощью команды CREATE TABLE. Эта команда создает пустую таблицу, то есть таблицу, не имеющую строк. Значения в эту таблицу вводятся с помощью команды INSERT. Команда CREATE TABLE определяет имя таблицы и множество поименованных столбцов в указанном порядке. Для каждого столбца должен быть определен тип и размер. Каждая создаваемая таблица должна иметь, по крайней мере, один столбец. Синтаксис команды CREATE TABLE имеет следующий вид:

```
CREATE TABLE <ИМЯ ТаблицЫ>
(<имя столбцаХтип данных>[(<размер>)]);
```

Используемые в SQL типы данных как минимум поддерживают стандарты ANSI *(American National Standards Institute — Американский национальный институт стандартов)* (см. раздел 1.5 «Типы данных SQL»):

```
CHAR (CHARACTER),
INT  (INTEGER),
SMALLINT,
DEC  (DECIMAL),
NUMERIC,
FLOAT,
```

Тип данных, для которого обязательно должен быть указан размер, — это CHAR. Реальное количество символов, которое

может находиться в поле, изменяется от нуля (если в поле содержится NULL-значение) до заданного в CREATE TABLE максимального значения.

Следующий пример показывает команду, которая позволяет создать таблицу STUDENT.

```
CREATE TABLE STUDENT1
 (STUDENT__ID INTEGER,
SURNAME      VARCHAR(60;,
NAME         VARCHAR (60),
STIPEND      DOUBLE,
KURS         INTEGER,
CITY         VARCHAR(60>,
BIRTHDAY     DATE,
UNIV_ID      INTEGER);
```

4.2. Использование индексации для быстрого доступа к данным

Операции поиска-выборки (SELECT) данных из таблиц по значениям их полей могут быть существенно ускорены путем использования индексации данных. Индекс содержит упорядоченный (в алфавитном или числовом порядке) список содержимого столбцов или группы столбцов в индексируемой таблице с идентификаторами этих строк (ROWID), Для пользователей индексирование таблицы по тем или иным столбцам представляет собой способ *логического* упорядочения значений индексированных столбцов, позволяющего, в отличие от последовательного перебора строк, существенно повысить скорость доступа к конкретным строкам таблицы при выборках, использующих значения этих столбцов. Индексация позволяет находить содержащий индексированную строку блок данных, выполняя небольшое число обращений к внешнему устройству.

При использовании индексации следует, однако, иметь в виду, что управление индексом существенно замедляет время выполнения операций, связанных с обновлением данных (та-

ких, как INSERT и DELETE), так как эти операции требуют перестройки индексов.

Индексы можно создавать как по одному, так и по множеству полей. Если указано более одного поля для создания единственного индекса, данные упорядочиваются по значениям первого поля, по которому осуществляется индексирование. Внутри получившейся группы осуществляется упорядочение по значениям второго поля, для получившихся в результате групп осуществляется упорядочение по значениям третьего поля и т.д.

Синтаксис команды создания индекса имеет следующий вид:

CREATE **INDEX <имя индекса> ON <имя табпицы>** (<имя столбца>
[,<имя столбца>]);

При этом таблица должна быть уже создана и содержать столбцы, имена которых указаны в команде создания индекса. Имя индекса, определенное в команде, должно быть уникальным в базе данных. Будучи однажды созданным, индекс является невидимым для пользователя, все операции с ним осуществляет СУБД.

Пример

Если таблица EXAM_MARKS часто используется для поиска оценки конкретного студента по значению поля STUDENT_ID, то следует создать индекс по этому полю.

```
CREATE INDEX STUDENT_ID_1 ON EXAM_MARKS (STUDENT_ID);
```

Для удаления индекса (при этом обязательно требуется знать его имя) используется команда DROP INDEX, имеющая следующий синтаксис:

DROP INDEX **<имя индекса>;**

Удаление индекса не изменяет содержимого поля или полей, индекс которых удаляется.

4.3. Изменение существующей таблицы

Для модификации структуры и параметров существующей таблицы используется команда ALTER TABLE. Синтаксис ко-

манды ALTER TABLE для добавления столбцов в таблицу имеет вид

```
ALTER TABLE <ИМЯ таблицы> ADD (<ИМЯ СТОЛбца> <ТИП ДАННЫХ>
        <размер>);
```

По этой команде для существующих в таблице строк добавляется новый столбец, в который заносится NULL-значение. Этот столбец становится последним в таблице. Можно добавлять несколько столбцов, в этом случае их определения в команде ALTER TABLE разделяются запятой.

Возможно изменение описания столбцов. Часто это связано с изменением размеров столбцов, добавлением или удалением ограничений, накладываемых на их значения. Синтаксис команды в этом случае имеет вид

```
ALTER TABLE <ИМЯ ТЭбЛИЦЫ> MODIFY <ИМЯ СТОЛбЦЭ> <ТИП ДАННЫХ>
        <размер/точность >;
```

Следует иметь в виду, что модификация характеристик столбца может осуществляться не в любом случае, а с учетом следующих ограничений:

- изменение типа данных возможно только в том случае, если столбец пуст;
- для незаполненного столбца можно изменять размер/точность. Для заполненного столбца размер/точность можно увеличить, но нельзя понизить;
- ограничение NOT NULL может быть установлено, если ни одно значение в столбце не содержит NULL. Опцию NOT NULL всегда можно отменить;
- разрешается изменять значения, установленные по умолчанию.

4.4. Удаление таблицы

Чтобы удалить существующую таблицу, необходимо предварительно удалить все данные из этой таблицы, то есть сделать ее пустой. Таблица, имеющая строки, не может быть удалена. Синтаксис команды, осуществляющей удаление пустой таблицы, имеет следующий вид:

```
DROP TABLE <имя таблицы>;
```

Упражнения

1. Напишите команду CREATE TABLE для создания таблицы LECTURER.
2. Напишите команду CREATE TABLE для создания таблицы SUBJECT.
3. Напишите команду CREATE TABLE для создания таблицы UNIVERSITY.
4. Напишите команду CREATE TABLE для создания таблицы EXAM_MARKS.
5. Напишите команду CREATE TABLE для создания таблицы SUBJ_LECT.
6. Напишите команду, которая позволит быстро выбрать данные о студентах по курсам, на которых они учатся.
7. Создайте индекс, который позволит для каждого студента быстро осуществить поиск оценок, сгруппированных по датам.

4.5. Ограничения на множество допустимых значений данных

До сих пор рассматривалось только следующее ограничение — значения, вводимые в таблицу, должны иметь типы данных и размеры, совместимые с типами/размером данных столбцов, в которые эти значения вводятся (как определено в команде CREATE TABLE или ALTER TABLE). Описание таблицы может быть дополнено более сложными ограничениями, накладываемыми на значения, которые могут быть вставлены в столбец или группу столбцов. Ограничения (CONSTRAINTS) являются частью определения таблицы.

При создании (изменении) таблицы могут быть определены ограничения на вводимые значения. В этом случае SQL будет отвергать любое из них при несоответствии заданным критериям. Ограничения могут быть статическими, ограничивающими значения или диапазон значений, вставляемых в столбец (CHECK, NOT NULL). Они могут иметь связь со всеми значениями столбца, ограничивая новые строки значениями, которые не содержатся в столбцах или их наборах (уникальные значения, первичные ключи). Ограничения могут также определяться связью со значениями, находящимися в другой таблице, допуская, например, вставку в столбец только тех значений, которые в данным *момент* содержатся также в другом столбце другой или этой же таблицы (внешний ключ). Эти ограничения носят динамический характер.

Существует два основных типа ограничений — ограничения на столбцы и ограничения на таблицу. Ограничения на столбцы (COLUMN CONSTRAINTS) применимы только к отдельным столбцам, а ограничения на таблицу (TABLE CONSTRAINTS) применимы к группам, состоящим из одного или более столбцов. Ограничения на столбец добавляются в конце определения столбца после указания типа данных и перед окончанием описания столбца (запятой). Ограничения на таблицу размещаются в конце определения таблицы, после определения последнего столбца. Команда CREATE TABLE имеет следующий синтаксис, расширенный включением ограничений:

```
CREATE TABLE <ИМЯ таблицы >

(<имя столбца > <тип данных> Ограничения на столбец>,

<имя столбца> <тип данных> Ограничения на столбец>,

Ограничения на таблицу> (<имя столбца>[,<имя столбца>]));
```

Поля, заданные в круглых скобках после описания ограничений таблицы, — это поля, на которые эти ограничения распространяются. Ограничения на столбцы применяются к тем столбцам, в которых они описаны.

4.5.1. Ограничение NOT NULL

Чтобы запретить возможность использования в поле NULL-значений, можно при создании таблицы командой CREATE TABLE указать для соответствующего столбца ключевое слово NOT NULL. Это ограничение применимо только к столбцам таблицы. Как уже говорилось выше, NULL — это специальный маркер, обозначающий тот факт, что поле пусто. Но он полезен не всегда. Первичные ключи, например, в принципе не должны содержать NULL-значений (быть пустыми), поскольку это нарушило бы требование уникальности первичного ключа (более строго — функциональную зависимость атрибутов таблицы от первичного ключа). Во многих других случаях также необходимо, чтобы поля *обязательно* содержали определенные значения. Если ключевое слово NOT NULL размещается непосредственно после типа данных (включая размер) столбца, то любые попытки оставить значение

поля пустым (ввести в поле NULL-значение) будут отвергнуты системой.

Например, для того, чтобы в определении таблицы STUDENT запретить использование NULL-значений для столбцов STUDENT_ID, SURNAME и NAME, можно записать следующее:

```
CREATE TABLE STUDENT
 (STUDENT_ID  INTEGER NOT NULL,
SURNAME      CHAR  (25)   NOT NULL,
NAME         CHAR  (10)   NOT NULL,
STIPEND       INTEGER,
KURS          INTEGER,
CITY          CHAR  (15),
BIRTHDAY      DATE,
UNIV_ID       INTEGER);
```

Важно помнить: если для столбца указано NOT NULL, то при использовании команды INSERT обязательно должно быть указано конкретное значение, вводимое в это поле. При отсутствии ограничения NOT NULL в столбце значение может отсутствовать, если только не указано значение столбца по умолчанию (DEFAULT). Если при создании таблицы ограничение NOT NULL не было указано, то его можно указать позже, используя команду ALTER TABLE. Однако для того, чтобы для вновь вводимого с помощью команды ALTER TABLE столбца можно было задать ограничение NOT NULL, таблица, в которую добавляется столбец, должна быть пустой.

4.5.2. Уникальность как ограничение на столбец

Иногда требуется, чтобы все значения, введенные в столбец, отличались друг от друга. Например, этого требуют первичные ключи. Если при создании таблицы для столбца указывается ограничение UNIQUE, то база данных отвергает любую попытку ввести в это поле какой-либо строки значение, уже содержащееся в том же поле другой строки. Это ограничение применимо только к тем полям, которые были объявлены NOT NULL. Можно

предложить следующее определение таблицы STUDENT, использующее ограничение UNIQUE:

```
CREATE TABLE STUDENT
 (STUDENT_ID INTEGER NOT NULL UNIQUE,
    SURNAME      CHAR (25)  NOT NULL,
    NAME         CHAR (10)  NOT NULL,
    STIPEND      INTEGER,
    KURS         INTEGER,
    CITY         CHAR (15),
    BIRTHDAY     DATE,
    UNIV_ID      INTEGER);
```

Объявляя поле STUDENT_ID уникальным, можно быть уверенным, что в таблице не появится записей для двух студентов *с* одинаковыми идентификаторами. Столбцы, отличные от первичного ключа, для которых требуется поддержать уникальность значений, называются возможными ключами или уникальными ключами (CANDIDATE KEYS ИЛИ UNIQUE KEYS).

4.5.3. Уникальность как ограничение таблицы

Можно сделать уникальными группу полей, указав UNIQUE в качестве ограничений *таблицы*. При объединении полей в группу важен порядок, в котором они указываются. Ограничение на таблицу UNIQUE является полезным, если требуется поддерживать уникальность группы полей. Например, если в нашей базе данных не допускается, чтобы студент сдавал в один день больше одного экзамена, то можно в таблице объявить уникальной комбинацию значений полей STUDENT_ID и EXAM_DATE. Для этого следует создать таблицу EXAM_MARKS следующим способом:

```
CREATE TABLE EXAM_MARKS
   (EXAM_ID     INTEGER NOT NULL,
   STUDENT_ID   INTEGER NOT NULL,
   SUBJ_ID      INTEGER NOT NULL,
   MARK         CHAR (1),
```

```
    EXAM_DATE    DATE NOT NULL,
    UNIQUE  (STUDENT_ID, EXAM_DATE));
```

Обратите внимание, что оба поля в ограничении таблицы UNIQUE все еще используют ограничение столбца — NOT NULL. Если бы использовалось ограничение столбца UNIQUE для поля STUDENT_ID, то такое ограничение таблицы было **бы** необязательным.

-Если значение поля STUDENT_ID должно быть различным для каждой строки в таблице EXAM_MARKS, это можно сделать, объявив UNIQUE как ограничение самого поля STUDENT_ID. В этом случае не будет и двух строк с идентичной комбинацией значений полей STUDENT_ID, EXAM_DATE. Следовательно, указание UNIQUE как ограничение таблицы наиболее полезно использовать в случаях, когда не требуется уникальность индивидуальных полей, как это имеет место на самом деле в рассматриваемом примере.

4,5.4. Присвоение имен ограничениям

Ограничениям таблиц можно присваивать уникальные имена. Преимущество явного задания имени ограничения состоит в том, что в этом случае при выдаче системой сообщения о нарушении установленного ограничения будет указано его имя, что упрощает обнаружение ошибок.

Для присвоения имени ограничению используется несколько измененный синтаксис команд CREATE TABLE и ALTER TABLE.

Приведенный выше пример запроса изменяется следующим образом:

```
CREATE TABLE EXAM_MARKS
    (EXAM_ID     INTEGER NOT NULL,
    STUDENT_ID   INTEGER NOT NULL,
    SUBJ_ID      INTEGER NOT NULL,
    MARK         CHAR (1),
    EXAM_DATE    DATE NOT NULL,
    CONSTRAINT   STUD_SUBJ_CONSTR
    UNIQUE       (STUDENT ID, EXAM DATE);
```

В этом запросе STUD__SUBJ_CONSTR — это имя, присвоенное указанному ограничению таблицы.

4.5.5. Ограничение первичных ключей

Первичные ключи таблицы — это специальные случаи комбинирования ограничений UNIQUE **и** NOT NULL. Первичные ключи имеют следующие особенности:

- таблица может содержать только один первичный ключ;
- внешние ключи по умолчанию ссылаются на первичный ключ таблицы;
- первичный ключ является идентификатором строк таблицы (строки, однако, могут идентифицироваться и другими способами).

Улучшенный вариант создания таблицы STUDENTI с объявленным первичным ключом имеет теперь следующий вид:

```
CREATE TABLE STUDENT
 (STUDENT_ID INTKGER PRIMARY KEY,
    SURNAME  CHAR  (25)  NOT NULL,
    NAME  CHAR  (10)  NOT NULL,
    STIPEND  INTEGER,
    KURS  INTEGER,
    CITY  CHAR  (15),
    BIRTHDAY DATE,
    UNIV_ID  INTEGER);
```

4.5.6. Составные первичные ключи

Ограничение PRIMARY KEY может также быть применено для нескольких полей, составляющих уникальную комбинацию значений — ***составной*** первичный ключ. Рассмотрим таблицу EXAM_MARKS. Очевидно, что ни к полю идентификатора студента (STUDENT_ID), ни к полю идентификатора предмета обучения (EXAM_ID) по отдельности нельзя предъявить требование уникальности. Однако для того, чтобы в таблице не могли появиться

разные записи для одинаковых комбинаций значений полей STUDENT_ID и EXAM_ID (конкретный студент на конкретном экзамене не может получить более одной оценки), имеет смысл объявить уникальной комбинацию этих полей. Для этого мы можем применить ограничение таблицы PRIMARY KEY объявив пару EXAM_ID И STUDENT_ID ПервИЧНЫМ КЛЮЧОМ Таблицы.

```
CREATE TABLE NEW_EXAM_MARKS
 (STUDENT_ID  INTEGER NOT NULL,
SUBJ_ID   INTEGER NOT NULL,
MARK    INTEGER,
DATA   DATE,
CONSTRAINT EX_PR_KEY PRIMARY KEY  (EXAM ю, STUDENT  ID));
```

4.5.7. Проверка значений полей

Ограничение CHECK позволяет определять условие, которому должно удовлетворять вводимое в поле таблицы значение, прежде чем оно будет принято. Любая попытка обновить или заменить значение поля такими, для которых предикат, задаваемый ограничением CHECK, имеет значение **ложь,** будет отвергаться.

Рассмотрим таблицу STUDENT. Значение столбца STIPEND в этой таблице выражается десятичным числом. Наложим на значения этого столбца ограничение — величина размера стипендии должна быть меньше 200.

Соответствующий запрос имеет следующий вид:

```
CREATE TABLE STUDENT
 (STUDENT_ID INTEGER PRIMARY KEY,
*  SURNAME      CHAR  (25)  NOT NULL,
NAME          CHAR  (10)  NOT NULL,
STIPEND      INTEGER CHECK  (STIPEND < 200),
KURS          INTEGER,
CITY          CHAR  (15),
BIRTHDAY      DATE,
UNIV_ID       INTEGER);
```

4.5.8. Проверка ограничивающих условий с использованием составных полей

Ограничение CHECK можно использовать в качестве табличного ограничения, то есть при необходимости включить более одного поля в ограничивающее условие.

Предположим, что ограничение на размер стипендии (меньше 200) должно распространяться только на студентов, живущих в Воронеже. Это можно указать в запросе со следующим табличным ограничением CHECK:

```
CREATE TABLE STUDENT
 (STUDENT_ID INTEGER PRIMARY KEY,
SURNAME     CHAR(25)  NOT NULL,
NAME        CHAR  (10)  NOT NULL,
STIPEND      INTEGER,
KURS         INTEGER,
CITY         CHAR (15),
BIRTHDAY     DATE,
UNIV_ID      INTEGER UNIQUE,
CHECK (STIPEND < 200 AND CITY =  'Воронеж'));
```

или в несколько другой записи:

```
CREATE TABLE STUDENT
   (STUDENT_ID  INTEGER PRIMARY KEY,
    SURNAME     CHAR(25)  NOT NULL,
    NAME        CHAR  (10)  NOT NULL,
    STIPEND      INTEGER,
    KURS         INTEGER,
    CITY         CHAR (15),
    BIRTHDAY     DATE,
    UNIV_ID      INTEGER UNIQUE,
  CONSTRAINT STUD_CHECK CHECK  (STIPEND < 200
AND CITY = 'Воронеж'));
```

4.5.9. Установка значений по умолчанию

В SQL имеется возможность при вставке в таблицу строки, не указывая значений некоторого поля, определять значение этого поля по умолчанию. Наиболее часто используемым значением по умолчанию является NULL. Это значение принимается по умолчанию для любого столбца, для которого не было установлено ограничение NOT NULL.

Значение поля по умолчанию указывается в команде CREATE TABLE тем же способом, что и ограничение столбца, с помощью ключевого слова

ОЕРАОТлКзначение по умолчанию>.

Строго говоря, опция DEFAULT не имеет ограничительного свойства, так как она не ограничивает значения, вводимые в поле, а просто конкретизирует значение поля в случае, если оно *не было задано*.

Предположим, что основная масса студентов, информация о которых находится в таблице STUDENT, проживает в Воронеже. Чтобы при задании атрибутов не вводить для большинства студентов название города 'Воронеж', можно установить его как значение поля CITY по умолчанию, определив таблицу STUDENT следующим образом:

```
CREATE TABLE STUDENT
    (STUDENT_ID  INTEGER PRIMARY KEY,
    SURNAME     CHAR  (25)  NOT NULL,
    NAME        CHAR  (10)  NOT NULL,
    STIPEND      INTEGER CHECK  (STIPEND <  200),
    KURS         INTEGER,
    CITY        CHAR  (15)  DEFAULT 'Воронеж',
    BIRTHDAY     DATE,
    UNIV_ID      INTEGER);
```

Другая цель практического применения задания значения по умолчанию — это использование его как альтернативы для NULL. Как уже отмечалось выше, присутствие NULL в качестве возможных значений поля существенно усложняет интерпретацию операций сравнения, в которых участвуют значения таких

полей, поскольку NULL представляет собой признак того, что фактическое значение поля *неизвестно* или *неопределенно*. Следовательно, строго говоря, сравнение с ним любого конкретного значения в рамках двузначной булевой логики является некорректным, за исключением специальной операции сравнения is NULL, которая определяет, является ли содержимое поля каким-либо значением или оно отсутствует. Действительно, каким образом в рамках двузначной логики ответить на вопрос, истинно или ложно условие CITY = 'Воронеж', если текущее значение поля CITY неизвестно (содержит NULL)?

Во многих случаях использование вместо NULL значения, подставляемого в поле по умолчанию, может существенно упростить использование значений поля в предикатах.

Например, можно установить для столбца опцию NOT NULL, а для неопределенных значений числового типа установить значение по умолчанию «равно нулю», или для полей типа CHAR пробел, использование которых **в** операциях сравнения не вызывает никаких проблем.

При использовании значений по умолчанию в принципе допустимо применять ограничения UNIQUE или PRIMARY KEY в этом поле. При этом, однако, следует иметь в виду отсутствие в таком ограничении практического смысла, поскольку только одна строка **в** таблице сможет принять значение, совпадающее с этим значением по умолчанию. Для такого применения задания по умолчанию необходимо (до вставки другой строки с установкой по умолчанию) модифицировать предыдущую строку, содержащую такое значение.

Упражнения

1. Создайте таблицу EXAMJVIARKS так, чтобы не допускался ввод в таблицу двух записей об оценках одного студента по конкретным экзамену и предмету обучения и чтобы не допускалось проведение двух экзаменов по любым предметам в один день,
2. Создайте таблицу предметов обучения SUBJECT так, чтобы количество отводимых на предмет часов по умолчанию было равно 36, не допускались записи с отсутствующим количеством часов, поле SUBJ_ID являлось первичным ключом таблицы и значения семестров (поле SEMESTER) лежали в диапазоне от 1 до 12.

3. Создайте таблицу EXAM_MARKS таким образом, чтобы значения поля EXAM_ID были больше значений поля SUBJ_ID, а значения поля SUBJ_ID были больше значений поля STUDENT_ID; пусть также будут запрещены значения NULL в любом из этих трех полей.

4.6. Поддержка целостности данных

В таблицах рассматриваемой базы данных значения некоторых полей связаны друг с другом. Так, поле STUDENT_ID в таблице STUDENT и поле STUDENT_ID в таблице EXAM_MARKS связаны тем, что описывают одни и те же объекты, то есть содержат идентификаторы студентов, информация о которых хранится в базе. Более того, значения идентификаторов студентов, которые допустимы в таблице EXAM_MARKS, должны выбираться только из списка значений STUDENT_ID, фактически присутствующих в таблице STUDENT, то есть принадлежащих реально описанным в базе студентам. Аналогично, значения поля UNIV_ID таблицы STUDENT должны соответствовать идентификаторам университетов UNIV_ID, фактически присутствующим в таблице UNIVERSITY, а значения поля SUBJ_ID таблицы EXAM_MARKS должны соответствовать идентификаторам предметов обучения, фактически присутствующим в таблице SYBJECT.

Ограничения, накладываемые указанным типом связи, называются *ограничениями ссылочной целостности*. Они составляют важную часть описания характеристик предметной области, обеспечения корректности данных, хранящихся в таблицах. Команды описания таблиц DML имеют средства, позволяющие описывать ограничения ссылочной целостности и обеспечивать поддержание такой целостности при манипуляциях значениями полей базы данных.

4.6.1. Внешние и родительские ключи

Когда каждое значение, присутствующее в одном поле таблицы, представлено в другом поле другой или этой же таблицы, говорят, что первое поле ссылается на второе. Это указывает на прямую связь между значениями двух полей. Поле, которое ссылается на другое поле, называется ***внешним ключом,*** а поле,