Выборка данных с использованием предложения SELECT

SELECT -

команда запроса, который получает любое количество данных из одного или нескольких отношений.

Результат запроса – другое отношение.

Предложение SELECT может использоваться как:

- самостоятельная команда на получение и вывод кортежей отношения;
- в качестве вложенного подзапроса;
- фраза выбора в командах определения схемы данных и модификации отношений;
- средство присвоения глобальным переменным значений кортежей сформированного отношения (INTO-фраза).

Упрощенный синтаксис SELECT

SELECT [**DISTINCT**] <список атрибутов>
FROM <список таблиц>
[**WHERE** <условие выборки>]
[**ORDER BY** < список атрибутов >]
[**GROUP BY** < список атрибутов >]
[**HAVING** <условие>]
[**UNION** <выражение с оператором **SELECT** >];

обязательно в конце!

SELECT – запрос на извлечение информации;

<список атрибутов> - наименование полей, содержимое которых запрашивается (через запятую);

FROM – обязательное ключевое слово;

<список таблиц> - имена таблиц, из которых извлекается информация (через запятую).

В [ ] необязательные элементы в запросе.

Есть таблица STUDENT:

STUDENT (Студент)							
STUDENT_ID	SURNAME	NAME	STIPEND	KURS	CITY	BIRTHDAY	UNIV_ID
1	Иванов	Иван	150	1	Орел	03.12.1982	10
3	Петров	Петр	200	3	Курск	01.12.1980	10
6	Сидоров	Вадим	150	4	Москва	07.06.1979	22
10	Кузнецов	Борис	0	2	Брянск	08.12.1981	10
12	Зайцева	Ольга	250	2	Липецк	01.05.1981	10
265	Павлов	Андрей	0	3	Воронеж	05.11.1979	10
32	Котов	Павел	150	5	Белгород	NULL	14
654	Лукин	Артем	200	3	Воронеж	01.12.1981	10
276	Петров	Антон	200	4	NULL	05.08.1981	22
55	Белкин	Вадим	250	5	Воронеж	07.01.1980	10

STUDENT_ID – числовой код, идентифицирующий студента;
SURNAME – фамилия студента;
NAME – имя студента;
STIPEND – стипендия, которую получает студент;
KURS – курс, на котором учиться студент;
CITY – город, в котором живет студент;
BIRTHDAY – дата рождения студента;
UNIV_ID – числовой код, идентифицирующий университет, в котором учиться студент.

Пример 1:

```
SELECT NAME, SURNAME
FROM STUDENT;
```

STUDENT (Студент)							
STUDENT_ID	SURNAME	NAME	STIPEND	KURS	CITY	BIRTHDAY	UNIV_ID
1	Иванов	Иван	150	1	Орел	03.12.1982	10
3	Петров	Петр	200	3	Курск	01.12.1980	10
6	Сидоров	Вадим	150	4	Москва	07.06.1979	22
10	Кузнецов	Борис	0	2	Брянск	08.12.1981	10
12	Зайцева	Ольга	250	2	Липецк	01.05.1981	10
265	Павлов	Андрей	0	3	Воронеж	05.11.1979	10
32	Котов	Павел	150	5	Белгород	NULL	14
654	Лукин	Артем	200	3	Воронеж	01.12.1981	10
276	Петров	Антон	200	4	NULL	05.08.1981	22
55	Белкин	Вадим	250	5	Воронеж	07.01.1980	10

Пример 1:

```
SELECT NAME, SURNAME
FROM STUDENT;
```

Результат запроса:

NAME	SURNAME
Иван	Иванов
Петр	Петров
Вадим	Сидоров
Борис	Кузнецов
Ольга	Зайцева
Андрей	Павлов
Павел	Котов
Артем	Лукин
Антон	Петров
Вадим	Белкин

Осуществляет выборку всех значений полей NAME и SURNAME из таблицы STUDENT.
Порядок следования столбцов в этой таблице соответствует порядку полей, указанному в запросе.

Пример 2:

Вывод значений всех столбцов таблицы

```
SELECT *
FROM STUDENT;
```

STUDENT (Студент)							
STUDENT_ID	SURNAME	NAME	STIPEND	KURS	CITY	BIRTHDAY	UNIV_ID
1	Иванов	Иван	150	1	Орел	03.12.1982	10
3	Петров	Петр	200	3	Курск	01.12.1980	10
6	Сидоров	Вадим	150	4	Москва	07.06.1979	22
10	Кузнецов	Борис	0	2	Брянск	08.12.1981	10
12	Зайцева	Ольга	250	2	Липецк	01.05.1981	10
265	Павлов	Андрей	0	3	Воронеж	05.11.1979	10
32	Котов	Павел	150	5	Белгород	NULL	14
654	Лукин	Артем	200	3	Воронеж	01.12.1981	10
276	Петров	Антон	200	4	NULL	05.08.1981	22
55	Белкин	Вадим	250	5	Воронеж	07.01.1980	10

В данном случает результатом выполнения будет таблица STUDENT полностью.

Пример 3:

```
SELECT CITY
FROM STUDENT;
```

STUDENT (Студент)							
STUDENT_ID	SURNAME	NAME	STIPEND	KURS	CITY	BIRTHDAY	UNIV_ID
1	Иванов	Иван	150	1	Орел	03.12.1982	10
3	Петров	Петр	200	3	Курск	01.12.1980	10
6	Сидоров	Вадим	150	4	Москва	07.06.1979	22
10	Кузнецов	Борис	0	2	Брянск	08.12.1981	10
12	Зайцева	Ольга	250	2	Липецк	01.05.1981	10
265	Павлов	Андрей	0	3	Воронеж	05.11.1979	10
32	Котов	Павел	150	5	Белгород	NULL	14
654	Лукин	Артем	200	3	Воронеж	01.12.1981	10
276	Петров	Антон	200	4	NULL	05.08.1981	22
55	Белкин	Вадим	250	5	Воронеж	07.01.1980	10

Пример 3:

```
SELECT CITY
FROM STUDENT;
```

Результат запроса:

CITY
Орел
Курск
Москва
Брянск
Липецк
Воронеж
Белгород
Воронеж
NULL
Воронеж

В полученной таблице несколько раз повторяется значение поля CITY: Воронеж.

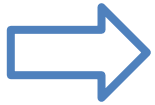
Для того, чтобы избежать повторяющихся записей используется команда DISTINCT.

Получаемые в результате выполнения **SQL**-запроса таблицы не в полной мере отвечают определению реляционного отношения: в них могут оказаться кортежи с одинаковым значением атрибутов.

Для исключения из результата **SELECT**-запроса повторяющихся записей используется ключевое слово **DISTINCT**.

Если запрос **SELECT** извлекает множество полей, то **DISTINCT** исключает дубликаты строк, в которых значения всех выбранных строк идентичны.

Предмет
БД и СУБД
БД и СУБД
БД и СУБД
БД и СУБД



Предмет
БД и СУБД

Пример 3 (*переделанный Пример 2*):

```
SELECT DISTINCT CITY
FROM STUDENT;
```

Результат запроса:

CITY
Орел
Курск
Москва
Брянск
Липецк
Воронеж
Белгород
NULL

С помощью DISTINCT избавились от повторяющихся записей.

Вместо **DISTINCT** может использоваться ключевое слово **ALL** для вывода повторяющихся. Режим, задаваемый ключевым словом **ALL**, действует по умолчанию, поэтому в реальных запросах для этих целей оно практически не используется.

Использование в операторе **SELECT** предложения **WHERE** (где), позволяет задавать выражение условия (предикат), принимающее значение истина или ложь.

Предложение **WHERE** определяет, какие строки должны быть выбраны. В таблицу, являющуюся результатом запроса, включаются только те строки, в которых **WHERE** принимает значение истина.

Пример 4:

```
SELECT NAME, SURNAME
FROM STUDENT
WHERE SURNAME='Петров';
```

STUDENT (Студент)

STUDENT_ID	SURNAME	NAME	STIPEND	KURS	CITY	BIRTHDAY	UNIV_ID
1	Иванов	Иван	150	1	Орел	03.12.1982	10
3	Петров	Петр	200	3	Курск	01.12.1980	10
6	Сидоров	Вадим	150	4	Москва	07.06.1979	22
10	Кузнецов	Борис	0	2	Брянск	08.12.1981	10
12	Зайцева	Ольга	250	2	Липецк	01.05.1981	10
265	Павлов	Андрей	0	3	Воронеж	05.11.1979	10
32	Котов	Павел	150	5	Белгород	NULL	14
654	Лукин	Артем	200	3	Воронеж	01.12.1981	10
276	Петров	Антон	200	4	NULL	05.08.1981	22
55	Белкин	Вадим	250	5	Воронеж	07.01.1980	10

Пример 4:

```
SELECT NAME, SURNAME
FROM STUDENT
WHERE SURNAME='Петров';
```

Результат запроса:

NAME	SURNAME
Петр	Петров

В задаваемых в предложении **WHERE** условиях могут использоваться операции сравнения, а также логические операторы **AND**, **OR**, **NOT**.

Пример 5:

```
SELECT NAME, SURNAME
FROM STUDENT
WHERE KURS=3 AND STIPEND>0;
```

STUDENT (Студент)							
STUDENT_ID	SURNAME	NAME	STIPEND	KURS	CITY	BIRTHDAY	UNIV_ID
1	Иванов	Иван	150	1	Орел	03.12.1982	10
3	Петров	Петр	200	3	Курск	01.12.1980	10
6	Сидоров	Вадим	150	4	Москва	07.06.1979	22
10	Кузнецов	Борис	0	2	Брянск	08.12.1981	10
12	Зайцева	Ольга	250	2	Липецк	01.05.1981	10
265	Павлов	Андрей	0	3	Воронеж	05.11.1979	10
32	Котов	Павел	150	5	Белгород	NULL	14
654	Лукин	Артем	200	3	Воронеж	01.12.1981	10
276	Петров	Антон	200	4	NULL	05.08.1981	22
55	Белкин	Вадим	250	5	Воронеж	07.01.1980	10

Пример 5:

```
SELECT NAME, SURNAME
FROM STUDENT
WHERE KURS=3 AND STIPEND>0;
```

Результат запроса:

NAME	SURNAME
Петр	Петров
Артем	Лукин

Запрос для получения имен и фамилий студентов, обучающихся на третьем курсе и получающих стипендию.

Операторы **IN** и **NOT IN**

IN – равен любому из списка
NOT IN – не равен ни одному из списка

Используются для сравнения проверяемого значения поля с заданным списком. Этот список значений указывается в скобках справа от оператора **IN**.

Построенный с использованием **IN** предикат, считается истинным, если значение поля, имя которого указано слева от **IN**, совпадает *(подразумевается точное совпадение)* с одним из значений, перечисленных в списке.

Предикат, построенный с использованием **NOT IN,** считается истинным, если значение поля не совпадает ни с одним из значений , перечисленных в списке.

Есть таблица EXAM_MARKS:

EXAM_MARKS (Экзаменационные оценки)				
EXAM_ID	STUDENT_ID	SUBJ_ID	MARK	EXAM_DATE
145	12	10	5	12.12.2000
34	32	10	4	23.01.2000
75	55	10	5	05.01.2000
238	12	22	3	17.06.1999
639	55	22	NULL	22.06.1999
43	6	22	4	18.01.2000

EXAM_ID – идентификатор экзамена;
STUDENT_ID – идентификатор студента;
SUBJ_ID – идентификатор предмета обучения;
MARK – экзаменационная оценка;
EXAM_DATE – дата экзамена.

Пример 1:

```
SELECT *
FROM EXAM_MARKS
WHERE MARK IN(4,5);
```

EXAM_MARKS
(Экзаменационные оценки)

EXAM_ID	STUDENT_ID	SUBJ_ID	MARK	EXAM_DATE
145	12	10	5	12.12.2000
34	32	10	4	23.01.2000
75	55	10	5	05.01.2000
238	12	22	3	17.06.1999
639	55	22	NULL	22.06.1999
43	6	22	4	18.01.2000

Результат запроса:

EXAM_ID	STUDENT_ID	SUBJ_ID	MARK	EXAM_DATE
145	12	10	5	12.12.2000
34	32	10	4	23.01.2000
75	55	10	5	05.01.2000
43	6	22	4	18.01.2000

Пример 2:

```
SELECT *
FROM EXAM_MARKS
WHERE MARK NOT IN(4,5);
```

EXAM_MARKS
(Экзаменационные оценки)

EXAM_ID	STUDENT_ID	SUBJ_ID	MARK	EXAM_DATE
145	12	10	5	12.12.2000
34	32	10	4	23.01.2000
75	55	10	5	05.01.2000
238	12	22	3	17.06.1999
639	55	22	NULL	22.06.1999
43	6	22	4	18.01.2000

Результат запроса:

EXAM_ID	STUDENT_ID	SUBJ_ID	MARK	EXAM_DATE
238	12	22	3	17.06.1999
639	55	22	NULL	22.06.1999
43	6	22	4	18.01.2000

Оператор **BETWEEN**

Используется для проверки условия вхождения значения поля в заданный интервал, то есть вместо списка значений атрибута этот оператор задает границы его изменения.

Замечание! Граничные значения входят во множество значений, с которыми производится сравнение.

Есть таблица SUBJECT:

SUBJECT (Предмет)			
SUBJ_ID	SUBJ_NAME	HOUR	SEMESTER
10	Информатика	56	1
22	Физика	34	1
43	Математика	56	2
56	История	34	4
94	Английский	56	3
73	Физкультура	34	5

EXAM_ID – идентификатор экзамена;
STUDENT_ID – идентификатор студента;
SUBJ_ID – идентификатор предмета обучения;
MARK – экзаменационная оценка;
EXAM_DATE – дата экзамена.

Пример :

```
SELECT *
FROM SUBJECT
WHERE HOUR BETWEEN 30 AND 40;
```

SUBJECT (Предмет)			
SUBJ_ID	SUBJ_NAME	HOUR	SEMESTER
10	Информатика	56	1
22	Физика	34	1
43	Математика	56	2
56	История	34	4
94	Английский	56	3
73	Физкультура	34	5

Результат запроса:

SUBJ_ID	SUBJ_NAME	HOUR	SEMESTER
22	Физика	34	1
56	История	34	4
73	Физкультура	34	5

Оператор **LIKE**

Применим только к символьным полям.

Этот оператор просматривает строковые значения полей с целью определения, входит ли заданная в операторе LIKE строка (образец поиска) в символьную строку-значение проверяемого поля.

Для выборки строковых значений по заданному образцу подстроки можно применять шаблон искомого образца строки, использующий следующий символ:

- «_» - определяет возможность наличия в указанном месте одного любого символа;
- «%» - допускает присутствие в указанном месте проверяемой строки последовательности любых символов произвольной длины.

Пример :

```
SELECT *
FROM STUDENT
WHERE SURNAME LIKE 'С%';
```

STUDENT (Студент)							
STUDENT_ID	SURNAME	NAME	STIPEND	KURS	CITY	BIRTHDAY	UNIV_ID
1	Иванов	Иван	150	1	Орел	03.12.1982	10
3	Петров	Петр	200	3	Курск	01.12.1980	10
6	Сидоров	Вадим	150	4	Москва	07.06.1979	22
10	Кузнецов	Борис	0	2	Брянск	08.12.1981	10
12	Зайцева	Ольга	250	2	Липецк	01.05.1981	10
265	Павлов	Андрей	0	3	Воронеж	05.11.1979	10
32	Котов	Павел	150	5	Белгород	NULL	14
654	Лукин	Артем	200	3	Воронеж	01.12.1981	10
276	Петров	Антон	200	4	NULL	05.08.1981	22
55	Белкин	Вадим	250	5	Воронеж	07.01.1980	10

Оператор **LIKE**

В случае необходимости включения в образец самих символов «_» и «%» применяют так называемые *escape*-символы.

LIKE '__P' ESCAPE '_\'

В этом выражении символ '\' с помощью ключевого слова **ESCAPE** объявляется *escape*-символом. Первый символ «_» будет соответствовать, как и ранее, любому символу в проверяемой строке. Однако второй символ «_», следующий после символа «\», объявленного *escape*-символом, уже будет интерпретироваться буквально как обычный символ.

IS NULL используется для проверки на пустое значение.

Пример :

```
SELECT *
FROM STUDENT
WHERE CITY IS NULL;
```

STUDENT (Студент)							
STUDENT_ID	SURNAME	NAME	STIPEND	KURS	CITY	BIRTHDAY	UNIV_ID
1	Иванов	Иван	150	1	Орел	03.12.1982	10
3	Петров	Петр	200	3	Курск	01.12.1980	10
6	Сидоров	Вадим	150	4	Москва	07.06.1979	22
654	Лукин	Артем	200	3	Воронеж	01.12.1981	10
276	Петров	Антон	200	4	NULL	05.08.1981	22
55	Белкин	Вадим	250	5	Воронеж	07.01.1980	10

Числовые и символьные константы

Несмотря на то, что SQL работает с данными в понятии строк и столбцов таблиц, имеется возможность применения значений выражений, построенных с использованием встроенных функций, констант, имен столбцов, определяемых как своего рода виртуальные столбцы. Они помещаются в списке столбцов и могут сопровождаться псевдонимами.

Если в запросе вместо спецификации столбца **SQL** обнаруживает число, то оно интерпретируется как числовая константа.

Символьные константы должны указываться в одинарных кавычках.

```
SELECT 'Фамилия', SURNAME, 'Имя', NAME, 100
FROM STUDENT;
```

Пример 1:

```
SELECT SURNAME, NAME, STIPEND,
-(STIPEND*KURS)/2
FROM STUDENT
WHERE KURS=4 AND STIPEND>0;
```

STUDENT (Студент)

STUDENT_ID	SURNAME	NAME	STIPEND	KURS	CITY	BIRTHDAY	UNIV_ID
1	Иванов	Иван	150	1	Орел	03.12.1982	10
3	Петров	Петр	200	3	Курск	01.12.1980	10
6	Сидоров	Вадим	150	4	Москва	07.06.1979	22
10	Кузнецов	Борис	0	2	Брянск	08.12.1981	10
12	Зайцева	Ольга	250	2	Липецк	01.05.1981	10
265	Павлов	Андрей	0	3	Воронеж	05.11.1979	10
32	Котов	Павел	150	5	Белгород	NULL	14
654	Лукин	Артем	200	3	Воронеж	01.12.1981	10
276	Петров	Антон	200	4	NULL	05.08.1981	22
55	Белкин	Вадим	250	5	Воронеж	07.01.1980	10

Пример 1 :

```
SELECT SURNAME, NAME, STIPEND,
-(STIPEND*KURS)/2
FROM STUDENT
WHERE KURS=4 AND STIPEND>0;
```

Результат запроса:

SURNAME	NAME	STIPEND	-(STIPEND*KURS)/2
Сидоров	Вадим	150	-300
Петров	Антон	200	-400

Можно использоваться следующие арифметические операции: унарный минус, +, -, *, /.

Пример 2 :

```
SELECT SURNAME || '_' || NAME, STIPEND
FROM STUDENT
WHERE KURS=4 AND STIPEND>0;
```

Результат запроса:

SURNAME_NAME	STIPEND
Иванов_Иван	150
Петров_Петр	200
Сидоров_Вадим	150
Кузнецов_Борис	0
Зайцева_Ольга	250
Павлов_Андрей	0
Котов_Павел	150
Лукин_Артем	200
Петров_Антон	200
Белкин_Вадим	250

Операция конкатенации «||» позволяет соединять значения двух или более столбцов символьного типа или символьных констант в одну строку.

В СУБД Access для этой цели в запросах SQL применяется символ «&».

Агрегирование и групповые функции

Позволяют получать из таблицы сводную (агрегированную) информацию, выполняя операции над группой строк таблицы.

- **COUNT** – определяет количество строк или значений поля и не являющихся NULL-значениями;
- **SUM** – вычисляет арифметическую сумму;
- **AVG** – вычисляет среднее значение;
- **MAX** – вычисляет наибольшее из значений;
- **MIN** – вычисляет наименьшее из значений;

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

Пример 1 :

```
SELECT AVG(MARK), COUNT(*)
FROM EXAM_MARKS;
```

EXAM_MARKS
(Экзаменационные оценки)

EXAM_ID	STUDENT_ID	SUBJ_ID	MARK	EXAM_DATE
145	12	10	5	12.12.2000
34	32	10	4	23.01.2000
75	55	10	5	05.01.2000
238	12	22	3	17.06.1999
639	55	22	NULL	22.06.1999
43	6	22	4	18.01.2000

COUNT

SUM/n

Результат запроса:

AVG(MARK)	COUNT(*)
10,5	6

GROUP BY

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

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

Замечание! В **GROUP BY** должны быть указаны все выбираемые столбцы, приведенные после ключевого слова **SELECT,** кроме столбцов, указанных в качестве аргумента в агрегирующей функции.

Пример 2 :

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

EXAM_MARKS (Экзаменационные оценки)				
EXAM_ID	STUDENT_ID	SUBJ_ID	MARK	EXAM_DATE
145	12	10	5	12.12.2000
34	32	10	4	23.01.2000
75	55	10	5	05.01.2000
238	12	22	3	17.06.1999
639	55	22	NULL	22.06.1999
43	6	22	4	18.01.2000

Результат запроса:

STUDENT_ID	MAX(MARK)
6	4
12	5
32	4
55	5

Пример 3 :

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

EXAM_MARKS
(Экзаменационные оценки)

EXAM_ID	STUDENT_ID	SUBJ_ID	MARK	EXAM_DATE
145	12	10	5	12.12.2000
34	32	10	4	23.01.2000
75	55	10	5	05.01.2000
238	12	22	3	17.06.1999
639	55	22	NULL	22.06.1999
43	6	22	4	18.01.2000

Пример 3 :

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

Результат запроса:

STUDENT_ID	SUBJ_ID	MAX(MARK)
6	22	4
12	10	5
12	22	3
32	10	4
55	10	5
55	22	

В данном случае строки вначале группируются по значениям первого столбца, а внутри этих групп - в подгруппы по значениям второго столбца.

Таким образом, предложение **GROUP BY** не только устанавливает столбцы, по которым осуществляется группирование, но и указывает порядок разбиения столбцов на группы.

Пример 4:

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

SUBJECT (Предмет)			
SUBJ_ID	SUBJ_NAME	HOUR	SEMESTER
10	Информатика	56	1
22	Физика	34	1
43	Математика	56	2
56	История	34	4
94	Английский	56	3
73	Физкультура	34	5

Поскольку группы записей с количеством часов больше 72 нет, то результатом выполнения этого запроса будет пустая таблица.

Пустые значения в агрегирующих функциях

Если аргументом функции **COUNT** является столбец, содержащий пустое значение, то **COUNT** вернет число строк, которые не содержат пустые значения и к которым применено определенное в **COUNT** условие или группирование.

Поведение функции **COUNT(*)** не зависит от пустых значений. Она возвратит общее количество строк в таблице.

Функция **AVG** вычисляет среднее значение всех известных значений множества элементов, то есть эта функция подсчитывает сумму известных значений и делит ее на количество этих значений, а не на общее количество значений, среди которых могут быть **NULL**-значения. Если столбец состоит только из пустых значений, то функция возвратит **NULL**.

Условные операторы

При отсутствии пустых значений условные операторы возвращают либо **TRUE** либо **FALSE**. Если в столбце присутствуют пустые значения, то может быть возвращено третье значение: **UNKNOWN**.

Оператор OR:

- Если результат двух условий, объединенных **OR** известен, то применяются правила булевой логики;
- Если результат одного утверждения **TRUE**, а второго – неизвестен, то результат будет **TRUE**;
- Если результат одного утверждения **FALSE**, а второго – неизвестен, то результат будет неизвестен;
- Если результат обоих утверждений неизвестен, то и результат будет **UNKNOWN**.

Условные операторы

Оператор NOT:

Обычный унарный оператор **NOT** обращает оценку **FRUE** в **FALSE** и наоборот, но **NOT NULL** по прежнем у будет возвращать пустое значение **NULL**. Следует отличать проверку **IS NULL** (**IS NOT NULL**).

Оператор AND:

- Если результат двух условий, объединенных **AND** известен, то применяются правила булевой логики;
- Если результат одного из утверждений **UNKNOWN**, а второго – **TRUE**, то состояние неизвестного утверждения является определяющим (т.е. результат будет **UNKNOWN**);
- Если результат одного из утверждений **UNKNOWN**, а второго – **FALSE**, то результат будет **FALSE**;
- Если результат обоих утверждений неизвестен, то и результат будет **UNKNOWN**

ORDER BY

Записи в реляционной БД не упорядочены, однако в результате выполнения запроса, данные можно упорядочить.

Для этого используется оператор **ORDER BY**, который позволяет упорядочивать выводимые записи в соответствии со значениями одного или нескольких выбранных столбцов.

При этом можно задать возрастающую (**ASC**) или убывающую (**DESC**) последовательность сортировки.

Пример 1:

```
SELECT *
FROM SUBJECT
ORDER BY SUBJ_NAME;
```

Результат запроса:

SUBJECT (Предмет)

SUBJ_ID	SUBJ_NAME	HOUR	SEMESTER
94	Английский	56	3
10	Информатика	56	1
56	История	34	4
43	Математика	56	2
22	Физика	34	1
73	Физкультура	34	5

Пример 1(*по убыванию*):

```
SELECT *
FROM SUBJECT
ORDER BY SUBJ_NAME DESC;
```

Результат запроса:

SUBJ_ID	SUBJ_NAME	HOUR	SEMESTER
73	Физкультура	34	5
22	Физика	34	1
43	Математика	56	2
56	История	34	4
10	Информатика	56	1
94	Английский	56	3

Предыдущий запрос, упорядоченный по убыванию.

Пример 2:

```
SELECT *
FROM SUBJECT
ORDER BY SEMESTER, SUBJ_NAME;
```

Результат запроса:

SUBJ_ID	SUBJ_NAME	HOUR	SEMESTER
10	Информатика	56	1
22	Физика	34	1
43	Математика	56	2
94	Английский	56	3
56	История	34	4
73	Физкультура	34	5

Предложение **ORDER BY** может использоваться для упорядочения групп записей. При этом оператор **ORDER BY** в запросе всегда должен быть последним.

Вложенные запросы

SQL позволяет использовать одни запросы внутри других запросов, то есть вкладывать запросы друг в друга.

Алгоритм работы запроса SQL со связанным подзапросом:

- Выбирается строка из таблицы, имя которой указано во внешнем запросе.
- Выполняется подзапрос и полученное 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.

Вложенные запросы

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

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

STUDENT (Студент)

STUDENT_ID	SURNAME	NAME	STIPEND	KURS	CITY	BIRTHDAY	UNIV_ID
1	Иванов	Иван	150	1	Орел	03.12.1982	10
3	Перов	Петр	200	3	Курск	01.12.1980	10
6	Сидоров	Вадим	150	4	Москва	07.06.1979	22
10	Кузнецов	Борис	0	2	Брянск	08.12.1981	10
12	Зайцева	Ольга	250	2	Липецк	01.05.1981	10
265	Павлов	Андрей	0	3	Воронеж	05.11.1979	10
32	Котов	Павел	150	5	Белгород	NULL	14
654	Лукин	Артем	200	3	Воронеж	01.12.1981	10
276	Петров	Антон	200	4	NULL	05.08.1981	22
55	Белкин	Вадим	250	5	Воронеж	07.01.1980	10

EXAM_MARKS
(Экзаменационные оценки)

EXAM_ID	STUDENT_ID	SUBJ_ID	MARK	EXAM_DATE
145	12	10	5	12.12.2000
34	32	10	4	23.01.2000
75	55	10	5	05.01.2000
238	12	22	3	17.06.1999
639	55	22	NULL	22.06.1999
43	6	22	4	18.01.2000
102	276	22	5	12.12.2000

Пример:
Известна фамилия студента (Петров), но неизвестно значение его идентификатора.

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

Результат запроса:

EXAM_ID	STUDENT_ID	SUBJ_ID	MARK	EXAM_DATE
102	276	22	5	12.12.2000

Пример 2 :

Данные обо всех оценках студентов из Воронежа.

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

EXAM_MARKS (Экзаменационные оценки)				
EXAM_ID	STUDENT_ID	SUBJ_ID	MARK	EXAM_DATE
145	12	10	5	12.12.2000
34	32	10	4	23.01.2000
75	55	10	5	05.01.2000
238	12	22	3	17.06.1999
639	55	22	NULL	22.06.1999
43	6	22	4	18.01.2000
102	276	22	5	12.12.2000

Пример 2 :
Данные обо всех оценках студентов из Воронежа.

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

Результат запроса:

EXAM_ID	STUDENT_ID	SUBJ_ID	MARK	EXAM_DATE
75	55	10	5	05.01.2000
639	55	22	NULL	22.06.1999

Пример 3:
Определить количество предметов обучения с оценкой, превышающей среднее значение оценки студента с идентификатором 55.

```
SELECT COUNT(SUBJ_ID), MARK
FROM EXAM_MARKS
GROUP BY MARK
HAVING MARK > (SELECT AVG(MARK)
        FROM EXAM_MARKS
        WHERE STUDENT_ID = 55);
```

EXAM_MARKS
(Экзаменационные оценки)

EXAM_ID	STUDENT_ID	SUBJ_ID	MARK	EXAM_DATE
145	12	10	5	12.12.2000
34	32	10	4	23.01.2000
75	55	10	5	05.01.2000
238	12	22	3	17.06.1999
639	55	22	NULL	22.06.1999
43	6	22	4	18.01.2000
102	276	22	5	12.12.2000

2

5:2=2,5

Пример 3:
Определить количество предметов обучения с оценкой, превышающей среднее значение оценки студента с идентификатором 55.

```
SELECT COUNT(SUBJ_ID), MARK
FROM EXAM_MARKS
GROUP BY MARK
HAVING MARK > (SELECT AVG(MARK)
        FROM EXAM_MARKS
        WHERE STUDENT_ID = 55);
```

Результат запроса:

COUNT(SUBJ_ID)	MARK
2	5

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 1:
Выбрать сведения обо всех предметах обучения, по которым проводился экзамен 20 января 1999 года.

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

EXAM_MARKS
(Экзаменационные оценки)

EXAM_ID	STUDENT_ID	SUBJ_ID	MARK	EXAM_DATE
145	12	10	5	12.12.2000
34	32	10	4	23.01.2000
75	55	10	5	05.01.2000
238	12	22	3	17.06.1999
639	55	22	NULL	22.01.1999
43	6	22	4	18.01.2000
102	276	22	5	12.12.2000

Пример 1:
Выбрать сведения обо всех предметах обучения, по которым проводился экзамен 20 января 1999 года.

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

Результат запроса:

EXAM_ID	STUDENT_ID	SUBJ_ID	MARK	EXAM_DATE
639	55	22	NULL	22.01.1999

В некоторых СУБД для выполнения этого запроса может потребоваться преобразование значения даты в символьный тип. В приведенном запросе SU и EX являются псевдонимами (*алиасами*).

Эту же операцию можно решить при помощи соединения таблиц:

```
SELECT SU.SUBJ_ID, SUBJ_NAME, HOUR, SEMESTER
FROM SUBJECT AS SU, EXAM_MARKS AS EX
WHERE SU.SUBJ_ID = EX.SUBJ_ID AND EX.EXAM_DATE =
#20/01/1999#;
```

Пример 2:
Найти студентов, которые получают стипендию выше средней на курсе.

```
SELECT DISTINCT STUDENT_ID, SURNAME, NAME, STIPEND
FROM STUDENT AS E1
WHERE STIPEND> (SELECT AVG(STIPEND)
                FROM STUDENT AS E2
                WHERE E1.KURS = E2.KURS);
```

STUDENT (Студент)

STUDENT_ID	SURNAME	NAME	STIPEND	KURS	CITY	BIRTHDAY	UNIV_ID
1	Иванов	Иван	150	1	Орел	03.12.1982	10
3	Перов	Петр	200	3	Курск	01.12.1980	10
6	Сидоров	Вадим	150	4	Москва	07.06.1979	22
10	Кузнецов	Борис	0	2	Брянск	08.12.1981	10
12	Зайцева	Ольга	250	2	Липецк	01.05.1981	10
265	Павлов	Андрей	0	3	Воронеж	05.11.1979	10
32	Котов	Павел	150	5	Белгород	NULL	14
654	Лукин	Артем	200	3	Воронеж	01.12.1981	10
276	Петров	Антон	200	4	NULL	05.08.1981	22
55	Белкин	Вадим	250	5	Воронеж	07.01.1980	10

AVG(STIPEND) – средняя стипендия по курсам

1 курс: 150:1=150; 2 курс: (250+0):2=125; 3 курс: (200+0+200):3=133;

4 курс: (150+200):2=175

Пример:
Найти студентов, которые получают стипендию выше средней на курсе.

```
SELECT DISTINCT STUDENT_ID, SURNAME,
NAME, STIPEND
FROM STUDENT AS E1
WHERE STIPEND> (SELECT AVG(STIPEND)
        FROM STUDENT AS E2
                WHERE E1.KURS =
E2.KURS);
```

Результат запроса:

STUDENT_ID	SURNAME	NAME	STIPEND
3	Перов	Петр	200
12	Зайцева	Ольга	250
654	Лукин	Артем	200
276	Петров	Антон	200

Пример:
Найти студентов, которые получают стипендию выше средней на курсе.

```
SELECT DISTINCT STUDENT_ID, SURNAME, NAME, STIPEND
FROM STUDENT AS E1
WHERE STIPEND> (SELECT AVG(STIPEND)
                FROM STUDENT AS  E2
                    WHERE E1.KURS = E2.KURS);
```

Второй вариант этой же задачи:

```
SELECT DISTINCT STUDENT_ID, SURNAME, STIPEND
FROM STUDENT E1,
   (SELECT KURS, AVG(STIPEND) AS AVG_STIPEND
   FROM STUDENT E2
   GROUP BY E2.KURS) E3
WHERE E1.STIPEND>AVG_STIPEND ANDE1.KURS=E3.KURS;
```

Пример 3:

По данным из таблицы EXAM_MARKS определить сумму полученных студентами оценок, сгруппировав значения оценок по датам экзаменов и исключив те дни, когда число студентов было больше 10.

```
SELECT EXAM_DATE, SUM(MARK)
FROM EXAM_MARKS AS A
GROUP BY EXAM_DATE
HAVING 10 > (SELECT COUNT (MARK)
             FROM EXAM_MARKS AS B
                  WHERE A.EXAM_DATE = B.EXAM_DATE);
```

EXAM_MARKS
(Экзаменационные оценки)

EXAM_ID	STUDENT_ID	SUBJ_ID	MARK	EXAM_DATE
145	12	10	5	12.12.2000
34	32	10	4	23.01.2000
75	55	10	5	05.01.2000
238	12	22	3	17.06.1999
639	55	22	NULL	22.01.1999
43	6	22	4	18.01.2000
102	276	22	5	12.12.2000

Пример 3:

По данным из таблицы EXAM_MARKS определить сумму полученных студентами оценок, сгруппировав значения оценок по датам экзаменов и исключив те дни, когда число студентов было больше 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);
```

Результат запроса:

EXAM_DATE	SUM(MARK)
22.01.1999	NULL
17.06.1999	3
05.01.2000	5
18.01.2000	4
23.01.2000	4
12.12.2000	10

Предикат предложения **HAVING** оценивается не для каждой строки результата, а для каждой *группы* выходных записей, сформированных предложением **GROUP BY** внешнего запроса.

Оператор **EXISTS**

Генерирует значение истина или ложь. Используя подзапросы в качестве аргумента, этот оператор оценивает результат выполнения подзапроса как истинный, если этот подзапрос генерирует выходные данные. В противном случае результат подзапроса ложный. Оператор **EXISTS** не может принимать значения **UNKNOWN**.

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 значение удовлетворяет условию, он возвращает значение **ИСТИНА** внешнему запросу, который затем анализирует свою строку-кандидата.

Часто **EXISTS** применяется с оператором **NOT**. Тогда запрос интерпретируется, как «не существует».

Замечание! В подзапросе, указываемом в операторе **EXIXTS** нельзя применять агрегирующие функции.

Пример 1:
Из таблицы студентов извлечь строки для каждого студента, сдавшего более одного предмета.

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

STUDENT (Студент)

STUDENT_ID	SURNAME	NAME	STIPEND	KURS	CITY	BIRTHDAY	UNIV_ID
1	Иванов	Иван	150	1	Орел	03.12.1982	10
3	Перов	Петр	200	3	Курск	01.12.1980	10
6	Сидоров	Вадим	150	4	Москва	07.06.1979	22
10	Кузнецов	Борис	0	2	Брянск	08.12.1981	10
12	Зайцева	Ольга	250	2	Липецк	01.05.1981	10
265	Павлов	Андрей	0	3	Воронеж	05.11.1979	10
32	Котов	Павел	150	5	Белгород	NULL	14
654	Лукин	Артем	200	3	Воронеж	01.12.1981	10
276	Петров	Антон	200	4	NULL	05.08.1981	22
55	Белкин	Вадим	250	5	Воронеж	07.01.1980	10

Пример 1:
Из таблицы студентов извлечь строки для каждого студента, сдавшего более одного предмета.

```
SELECT *
FROM STUDENT AS F
WHERE EXISTS (SELECT SUBJ_ID
        FROM EXAM_MARKS AS S
        WHERE F.STUDENT_ID=S.STUDENT_ID
        GROUP BY SUBJ_ID
        HAVING COUNT(SUBJ_ID)>1);
```

Результат запроса:

STUD_ID	SURNAME	NAME	STIPEND	KURS	CITY	BIRTHDAY	UNIV_ID
12	Зайцева	Ольга	250	2	Липецк	01.05.1981	10
55	Белкин	Вадим	250	5	Воронеж	07.01.1980	10

Оператор **UNION**

Используется для объединения выходных данных двух или более **SQL**-запросов в единое множество строк и столбцов.

Использование оператора **UNION** возможно только при объединении запросов, соответствующие столбцы которых совместимы по объединению, то есть совпадающие 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.

Есть таблица LECTURER:

LECTURER (Преподаватель)				
LECTURER_ID	SURNAME	NAME	CITY	UNIV_ID
24	Колесников	Борис	Воронеж	10
46	Никонов	Иван	Воронеж	10
74	Лагутин	Павел	Москва	22
108	Струков	Николай	Москва	22
276	Николаев	Виктор	Воронеж	10
328	Сорокин	Андрей	Орел	10

LECTURER_ID – числовой код, идентифицирующий преподавателя;
SURNAME – фамилия преподавателя;
NAME – имя преподавателя;
CITY – город, в котором живет преподаватель;
BIRTHDAY – дата рождения студента;
UNIV_ID – числовой код, идентифицирующий университет, в котором преподает преподаватель.

Пример 1:
Получить в одной таблице фамилии и идентификаторы студентов и преподавателей из Москвы.

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

LECTURER (Преподаватель)

LECTURER_ID	SURNAME	NAME	CITY	UNIV_ID
24	Колесников	Борис	Воронеж	10
46	Никонов	Иван	Воронеж	10
74	Лагутин	Павел	Москва	22
108	Струков	Николай	Москва	22
276	Николаев	Виктор	Воронеж	10
328	Сорокин	Андрей	Орел	10

Пример:
Получить в одной таблице фамилии и идентификаторы студентов и преподавателей из Москвы.

```
SELECT ‘Студент’ AS STATUS, SURNAME, STUDENT_ID
FROM STUDENT
WHERE CITY = ‘Москва’
UNION
SELECT ‘Преподаватель ’ AS STATUS, SURNAME, LECTURER_ID
FROM LECTURER
WHERE CITY = ‘Москва’;
```

Результат запроса:

STATUS	SURNAME	STUDENT_ID
Студент	Сидоров	6
Преподаватель	Лагутин	74
Преподаватель	Струков	108

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 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**.

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

Еще один пример использования **UNION**. Необходимо составить отчет, содержащий для каждой даты сдачи экзаменов сведения по каждому студенту, получившему максимальную и минимальную оценки.

```
SELECT 'MAX MARK', 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 'MIN MARK', 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));
```

В приведенном запросе агрегирующие функции используются в подзапросах. Это является нетрадиционным с точки зрения времени, затрачиваемого на выполнение запроса. Более эффективна форма запроса, возвращающего аналогичный результат.

```
SELECT 'MAX MARK', 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 'MIN MARK', 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;
```

Предложение **ORDER BY** применяется для упорядочения выходных данных объединения запросов так же, как и для отдельных запросов.

Модифицировать предыдущий пример, добавив упорядочение по фамилиям студентов и датам экзаменов можно с помощью добавления последней строки в запрос

ORDER BY SURNAME, E.EXAM_DATE;

Есть таблица UIVERSITY:

UNIVERSITY (Университеты)			
UNIV_ID	UNIV_NAME	RATTING	CITY
22	МГУ	606	Москва
10	ВГУ	296	Воронеж
11	НГУ	345	Новосибирск
32	РГУ	416	Ростов
14	БГУ	326	Белгород
15	ТГУ	368	Томск
18	ВГМА	327	Воронеж

UNIV_ID – идентификатор университета;
UNIV_NAME – название университета;
RATTING – рейтинг университета;
CITY – город, в котором расположен университет.

Пример 1:

Составить список студентов с указанием наименования университета для тех студентов, у которых эти данные есть, но при этом не исключая и студентов, у которых университет не указан.

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

Связь с таблицей UNIVERSITY

STUDENT (Студент)

STUDENT_ID	SURNAME	NAME	STIPEND	KURS	CITY	BIRTHDAY	UNIV_ID
1	Иванов	Иван	150	1	Орел	03.12.1982	10
3	Перов	Петр	200	3	Курск	01.12.1980	10
6	Сидоров	Вадим	150	4	Москва	07.06.1979	22
10	Кузнецов	Борис	0	2	Брянск	08.12.1981	10
12	Зайцева	Ольга	250	2	Липецк	01.05.1981	
265	Павлов	Андрей	0	3	Воронеж	05.11.1979	
32	Котов	Павел	150	5	Белгород	NULL	14
654	Лукин	Артем	200	3	Воронеж	01.12.1981	10
276	Петров	Антон	200	4	NULL	05.08.1981	22
55	Белкин	Вадим	250	5	Воронеж	07.01.1980	10

Пример 1:
Составить список студентов с указанием наименования университета для тех студентов, у которых эти данные есть, но при этом не исключая и студентов, у которых университет не указан.

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

Результат запроса:

SURNAME	NAME	UNIV_NAME	SURNAME	NAME	
Иванов	Иван	ВГУ	Зайцева	Ольга	UNKNOWN
Перов	Петр	ВГУ	Павлов	Андрей	UNKNOWN
Сидоров	Вадим	МГУ			
Кузнецов	Борис	ВГУ			
Котов	Павел	БГУ			
Лукин	Артем	ВГУ			
Петров	Антон	МГУ			
Белкин	Вадим	ВГУ			

Оператор **JOIN**

Если в операторе **SELECT** после ключевого слова **FROM** указывается не одна, а две таблицы, то в результате выполнения запроса, в котором отсутствует предложение **WHERE**, каждая строка одной таблицы будет соединена с каждой строкой второй таблицы. Такая операция называется **декартовым произведением** или **полным соединением** таблиц.

Сама по себе эта операция не имеет практического значения, более того, при ошибочном использовании может привести к неожиданным нештатным ситуациям. Соединение таблиц имеет смысл, в случае, если объединяются не все строки исходных таблиц, а только нужные пользователю. Такое ограничение может быть осуществлено с применением предложения **WHERE**.

Пример 1:

Получить фамилии студентов и для каждого студента – названия университетов, расположенных в городе, где живет студент. Т.е. необходимо получить все комбинации записей о студентах и университетах в двух таблицах, в которых значение поля **CITY** совпадает.

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

STUDENT (Студент)

STUDENT_ID	SURNAME	CITY
1	Иванов	Орел
3	Перов	Курск
6	Сидоров	Москва
10	Кузнецов	Брянск
12	Зайцева	Липецк
265	Павлов	Воронеж
32	Котов	Белгород
654	Лукин	Воронеж
276	Петров	NULL
55	Белкин	Воронеж

UNIVERSITY (Университеты)

UNIV_ID	UNIV_NAME	CITY
22	МГУ	Москва
10	ВГУ	Воронеж
11	НГУ	Новосибирск
32	РГУ	Ростов
14	БГУ	Белгород
15	ТГУ	Томск
18	ВГМА	Воронеж

Пример 1:
Получить фамилии студентов и для каждого студента – названия университетов, расположенных в городе, где живет студент. Т.е. необходимо получить все комбинации записей о студентах и университетах в двух таблицах, в которых значение поля **CITY** совпадает.

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

Результат запроса:

SURNAME	UNIV_NAME	CITY
Сидоров	МГУ	Москва
Павлов	ВГМА	Воронеж
Лукин	ВГУ	Воронеж
Белкин	ВГУ	Воронеж
Павлов	ВГУ	Воронеж
Лукин	ВГМА	Воронеж
Белкин	ВГМА	Воронеж
Котов	БГУ	Белгород

Соединение, использующее предикаты, основанные на равенствах, называется **эквисоединением**.

Рассмотренный пример соединения таблиц относится к виду **внутреннего (INNER) соединения**. При этом соединяются только те строки, для которых истинным является предикат, задаваемые в предложении **ON** выполняемого запроса.

Приведенный выше пример 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 запросе **JOIN** действует по умолчанию.

Пример полного соединения таблиц с использованием **JOIN:**

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

В Access данный запрос не идет.

JOIN не поддерживается в Oracle.

Операции объединения таблиц

Информация из таблиц студентов и экзаменационных оценок связана посредством поля **STUDENT_ID**. Причем в данном соединении поддерживается требование *целостности по ссылкам* .

База данных обладает свойством ссылочной целостности, когда для любой пары связанных внешним ключом отношений в ней условие ссылочной целостности выполняется.

Если вышеприведённое условие не выполняется, говорят, что в базе данных *нарушена ссылочная целостность*. Такая БД не может нормально эксплуатироваться, так как в ней разорваны логические связи между зависимыми друг от друга фактами. Непосредственным результатом нарушения ссылочной целостности становится то, что корректным запросом не всегда удаётся получить корректный результат.

Пример 1:
Получить список фамилий студентов с полученными ими оценками и идентификаторами предметов.

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

С оператором **JOIN:**

```
SELECT SURNAME, MARK
FROM STUDENT INNER JOIN
EXAM_MARKS ON
STUDENT.STUDENT_ID =
EXAM_MARKS.STUDENT_ID;
```

Результат запроса:

SURNAME	MARK	SUBJ_ID
Зайцева	5	10
Котов	4	10
Белкин	5	10
Зайцева	3	22
Белкин	NULL	22
Сидоров	4	22
Петров	5	22

Пример 2:
Найти фамилии всех студентов, получивших неудовлетворительную оценку, вместе с названиями предметов, по которым получена эта оценка.

```
SELECT SUBJ_NAME, SURNAME,
MARK
FROM STUDENT, SUBJECT,
EXAM_MARKS
WHERE
STUDENT.STUDENT_ID=EXAM_MAR
KS.STUDENT_ID
AND SUBJECT.SUBJ_ID =
EXAM_MARKS.SUBJ_ID
AND EXAM_MARKS.MARK = 3;
```

С оператором **JOIN:**

```
SELECT SUBJ_NAME, SURNAME,
MARK
FROM STUDENT INNER JOIN
SUBJECT INNER JOIN
EXAM_MARKS ON
STUDENT.STUDENT_ID=EXAM_MA
RKS.STUDENT_ID AND
SUBJECT.SUBJ_ID =
EXAM_MARKX.SUBJ_ID AND
EXAM_MARKS.MARK = 3;
```

Результат запроса:

SURNAME	MARK	SUBJ_NAME
Зайцева	3	Физика

Внешнее соединение таблиц

Получить записи о студентах с полученной оценкой и идентификатором предмета, включая тех, которые еще не сдавали экзамены и которые следовательно отсутствуют в таблице **EXAM_MARKS**.

Левое внешнее соединение:

```
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.

Правое внешнее соединение:

```
SELECT SURNAME, MARK
FROM EXAM_MARKS RIGHT OUTER
JOIN STUDENT
ON STUDENT.STUDENT_ID =
EXAM_MARKS.STUDENT_ID;
```

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