

ЛАБОРАТОРНАЯ РАБОТ 4

МАНИПУЛИРОВАНИЕ ДАННЫМИ С ИСПОЛЬЗОВАНИЕМ ЯЗЫКА SQL. ОПЕРАТОР JOIN

Для выполнения данной лабораторной работы Вам понадобится установленная СУБД Microsoft SQL Server версии не ниже 2014. Процесс установки представлен в разделе 5 данного методического указания. Взаимодействие с СУБД Microsoft SQL Server осуществляется через графическую оболочку Microsoft SQL Server Management Studio.

Цель работы: Изучить возможности оператора JOIN, получить навыки формирования SQL запросов на соединение таблиц на примере созданной базы данных.

ЗАДАНИЕ

1. Ознакомиться с теоретическим разделом;
2. Для своих вариантов выполнить:
 - a. Два запроса с соединением INNER JOIN двух таблиц
 - b. Два запроса с соединением INNER JOIN трех таблиц
 - c. Один запрос с соединением INNER JOIN четырех и более таблиц
 - d. Один запрос для каждого из типов соединений: FULL OUTER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, CROSS JOIN
3. Отобразить созданные запросы в отчете с комментариями. Содержание отчета:
 - титульный лист;
 - цель работы;
 - задание;
 - описание хода выполнения работы (написать запросы и прикрепить скриншоты результатов работы по каждому запросу);
 - вывод;

1. ОПЕРАТОРЫ СОЕДИНЕНИЯ JOIN

Ранее мы рассмотрели применение инструкции SELECT для выборки данных из одной таблицы базы данных. Если бы возможности языка SQL ограничивались поддержкой только таких простых инструкций SELECT, то присоединение в запросе двух или больше таблиц для выборки из них данных было бы невозможно. Следственно, все данные базы данных требовалось бы хранить в одной таблице. Хотя такой подход является вполне возможным, ему присущ один значительный недостаток - хранимые таким образом данные характеризуются высокой избыточностью.

Для устранения этого недостатка необходим **оператор соединения JOIN**, который позволяет извлекать данные более чем из одной таблицы. Этот оператор является наиболее важным оператором для реляционных систем баз данных, поскольку благодаря ему имеется возможность распределять данные по нескольким таблицам, обеспечивая, таким образом, важное свойство систем баз данных - отсутствие избыточности данных.

Для соединения таблиц можно использовать две разные синтаксические формы оператора соединения:

- явный синтаксис соединения (синтаксис соединения ANSI SQL: 1992);
- неявный синтаксис соединения (синтаксис соединения «старого стиля»)

Синтаксис соединения ANSI SQL:1992 был введен стандартом SQL92 и определяет операции соединения явно, т.е. используя соответствующее имя для каждого типа операции соединения. При явном объявлении соединения используются следующие ключевые слова:

- **CROSS JOIN** – определяет декартово произведение двух таблиц т.е. при выводе будут присутствовать все возможные комбинации строк выбранных таблиц;
- **[INNER] JOIN** - определяет естественное соединение двух таблиц;
- **LEFT [OUTER] JOIN** – определяет операцию левого соединения;
- **RIGHT [OUTER] JOIN** – определяет операцию правого соединения;
- **FULL [OUTER] JOIN** - определяет соединение правого и левого внешнего соединений.

Неявный синтаксис оператора соединения является синтаксисом "старого стиля", где каждая операция соединения определяется неявно посредством предложения **WHERE**, используя так называемые столбцы соединения.

Для операций соединения рекомендуется использовать явный синтаксис (с использованием **JOIN**), т.к. это повышает надежность запросов.

1.1. Создание базы данных

Для выполнения лабораторной работы рассмотрим пример базы данных,

```
CREATE DATABASE TEST_JOIN ON PRIMARY
(
    NAME = 'TEST_JOIN',
    FILENAME = 'D:\TEST_JOIN.mdf' ,
    SIZE = 3072KB ,
    MAXSIZE = UNLIMITED,
    FILEGROWTH = 1024KB )
LOG ON
(
    NAME = 'TEST_JOIN_log',
    FILENAME = 'D:\TEST_JOIN_log.ldf' ,
    SIZE = 1024KB ,
    MAXSIZE = 2048GB ,
    FILEGROWTH = 10%
)
GO
```

Которая содержит следующие таблицы:

```
CREATE TABLE PERSON
(
    ID INTEGER IDENTITY(1,1),
    NAME VARCHAR(50) NOT NULL,
    CID INTEGER NOT NULL,
    PRIMARY KEY (ID),
);
GO

CREATE TABLE CITY
(
    CID INTEGER IDENTITY(1,1),
    CITYNAME VARCHAR(50) NOT NULL,
    PRIMARY KEY (CID),
);
GO
```

Заполним таблицы данными:

```
INSERT INTO PERSON VALUES
```

```
( 'Андрей', 1 ),  
( 'Леонид', 2 ),  
( 'Сергей', 1 ),  
( 'Григорий', 4 )
```

```
INSERT INTO CITY VALUES
```

```
( 'Москва' ),  
( 'Санкт-Петербург' ),  
( 'Казань' )
```

```
GO
```

В результате получилась следующая тестовая схема БД:

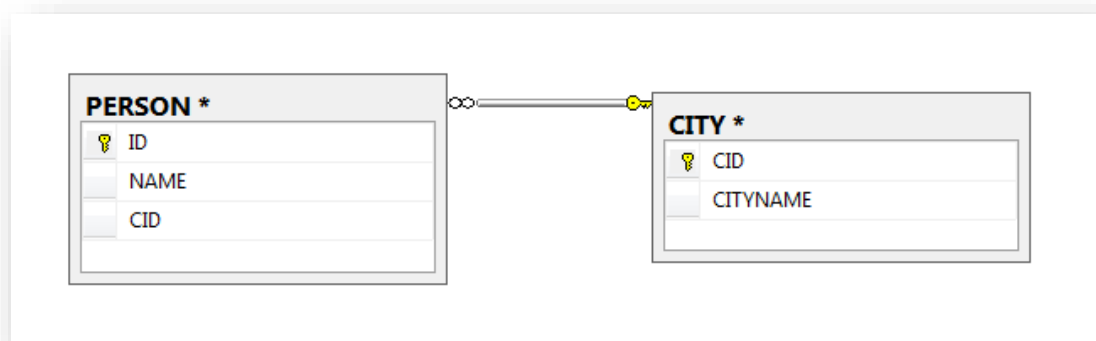


Рисунок 1 – Схема тестовой базы данных

1.2. Оператор внутреннего соединения INNER JOIN

Оператор внутреннего соединения INNER JOIN соединяет две таблицы. Порядок таблиц для оператора неважен, поскольку оператор является симметричным. Заголовок таблицы-результата является объединением заголовков соединяемых таблиц. Тело результата логически формируется следующим образом. Каждая строка одной таблицы сопоставляется с каждой строкой второй таблицы, после чего для полученной "соединённой" строки проверяется условие соединения (вычисляется предикат соединения). Если условие истинно, в таблицу-результат добавляется соответствующая "соединённая" строка.

Ключевое слово INNER можно опустить. Запись INNER JOIN идентична JOIN.

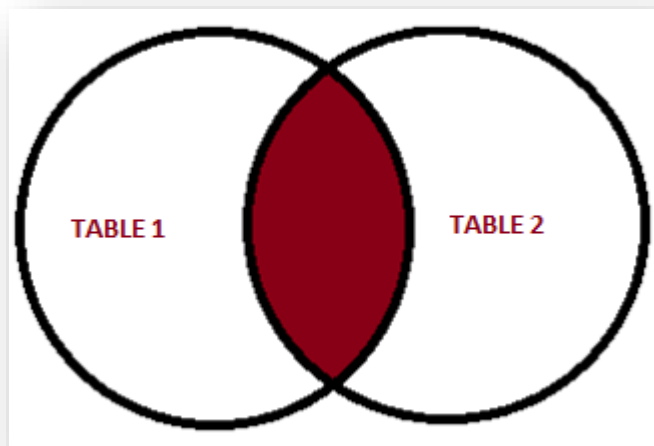


Рисунок 2 - INNER JOIN

Пример 1. Соединим две таблицы CITY и PERSON и выведем всю информацию о жителях и городе в котором они проживают.

```
SELECT CITY.*, PERSON.*  
FROM CITY INNER JOIN PERSON  
ON PERSON.CID = CITY.CID;
```

Получим следующий результат:

	CID	CITYNAME	ID	NAME	CID
1	1	Москва	1	Андрей	1
2	2	Санкт-Петербург	2	Леонид	2
3	1	Москва	3	Сергей	1

Рисунок 3 – Результат соединения

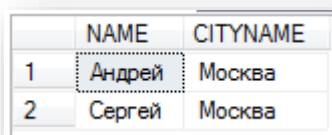
В этом примере в инструкции SELECT для выборки указаны все столбцы таблиц для жителя PERSON и города CITY. Предложение FROM инструкции SELECT определяет соединяемые таблицы, а также явно указывает тип операции соединения - INNER JOIN. Предложение ON является частью предложения FROM и указывает соединяемые столбцы в обеих таблицах. Выражение PERSON.CID = CITY.CID определяет условие соединения, а оба столбца условия называются столбцами соединения.

В инструкции SELECT с операцией соединения, кроме условия соединения предложение WHERE может содержать и другие условия, как это показано в следующем примере.

Пример 2. Соединим два таблицы CITY и PERSON и выведем имена жителей, которые живут в городе на букву М.

```
SELECT NAME, CITYNAME FROM PERSON  
INNER JOIN CITY ON PERSON.CID = CITY.CID  
WHERE CITYNAME LIKE 'M%'
```

Получим следующий результат:



	NAME	CITYNAME
1	Андрей	Москва
2	Сергей	Москва

Рисунок 4 – Результат запроса

Пример 3. Для более сложного запроса приведем пример из другой базы данных (БД библиотеки). Необходимо получить информацию о книгах (название, издательство, жанр, количество экземпляров в библиотеке), имеющих 2 и более авторов.

Часть схемы БД с необходимыми таблицами выглядит следующим образом:

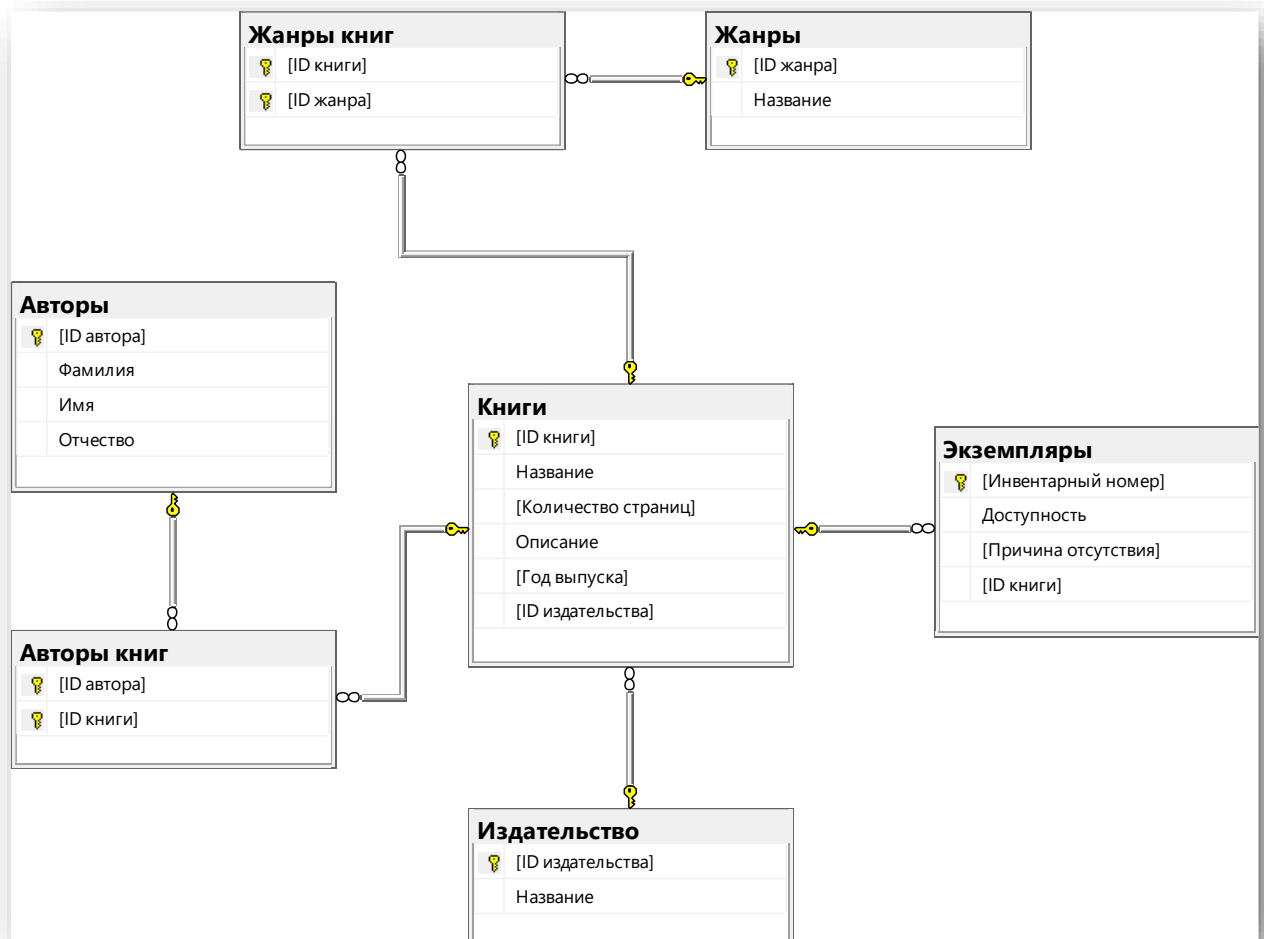


Рисунок 5 – Схема базы данных

Запрос для получения информации:

```

SELECT Книги.Название AS Книга,
       Издательство.Название AS Издательство,
       Жанры.Название AS Жанр,
       COUNT (DISTINCT Экземпляры.[Инвентарный номер]) AS
[Количество экземпляров]
FROM
(
  SELECT [Авторы книг].[ID книги], COUNT (*) AS [Количество авторов]
  FROM [Авторы книг]
  GROUP BY [Авторы книг].[ID книги]
  HAVING COUNT (*) >= 2
) КН INNER JOIN
  Книги ON КН.[ID книги] = Книги.[ID книги] INNER JOIN
  Издательство ON Книги.[ID издательства] = Издательство.[ID
издательства] INNER JOIN
  
```

```

[Жанры книг] ON Книги.[ID книги] = [Жанры книг].[ID книги] INNER
JOIN
Жанры ON [Жанры книг].[ID жанра] = Жанры.[ID жанра] INNER JOIN
Экземпляры ON Книги.[ID книги] = Экземпляры.[ID книги]
GROUP BY Книги.Название, Издательство.Название, Жанры.Название

```

Получим следующий результат:

Результаты		Сообщения		
	Книга	Издательство	Жанр	Количество экземпляров
1	1000 и 1 секрет BIOS по тонкой настройке, решению п...	Наука и техника	Компьютеры и Интернет	2
2	QNX/UNIX: Анатомия параллелизма	Символ-Плюс	Компьютеры и Интернет	3
3	Как тестируют в Google	Питер	Компьютеры и Интернет	2
4	Компьютерные сети	Форум	Компьютеры и Интернет	2
5	Компьютерные сети	Форум	Наука, Образование	2
6	Компьютерные сети. Принципы, технологии, протоколы	Питер Принт	Компьютеры и Интернет	2
7	Основы информатики	Новое знание	Компьютеры и Интернет	2
8	Основы информатики	Новое знание	Наука, Образование	2
9	Параллельное и распределенное программирование ...	Вильямс	Компьютеры и Интернет	2
10	Параллельное и распределенное программирование ...	Вильямс	Наука, Образование	2
11	Создание сайтов	Питер	Компьютеры и Интернет	2
12	Создание сайтов	Питер	Приключения	2
13	Теория систем и системный анализ	ТУСУР	Наука, Образование	1

Рисунок 6 – Результат запроса

Оператор GROUP BY группирует записи по указанным после оператора через запятую именам колонок. После оператора SELECT нужно перечислить те же имена колонок, что и для GROUP BY.

Функция COUNT() используется для подсчета количества строк в группе.

HAVING используется также как и WHERE, но после группировок.

В данном примере также используется подзапрос. Подзапросы, внутренние или вложенные запросы – есть не что иное, как запрос внутри запроса. Они обычно используются в конструкциях WHERE, SELECT, FORM.

В приведенном примере в секции FROM стоит подзапрос, заключенный в круглые скобки. Для каждого подзапроса в секции FROM указывается псевдоним (в примере это Кн), иначе невозможно работать с полями запросов. Получается, что вместо того, чтобы получить данные непосредственно из таблицы, мы получаем их из запроса.

Подзапрос в секции SELECT должен возвращать только одну строку. Если результатом будет несколько строк, то запрос возвращает ошибку.

При использовании подзапроса в конструкции WHERE результат должен состоять только из одной колонки. Это значит, что нельзя написать во внутреннем запросе

SELECT *, а можно только SELECT ИмяОдногоПоля. Имя должно быть только одно, и тип его должен совпадать с типом сравниваемого значения.

Подзапросы нужно использовать очень аккуратно, потому что они могут привести к ошибке.

1.3. Оператор внешнего соединения OUTER JOIN

В предшествующих примерах естественного соединения, результирующий набор содержал только те строки с одной таблицы, для которых имелись соответствующие строки в другой таблице. Но иногда кроме совпадающих строк бывает необходимым извлечь из одной или обеих таблиц строки без совпадений. Такая операция называется внешним соединением OUTER JOIN. Внешнее соединение может быть:

- Левым - **LEFT OUTER JOIN**
- Правым - **RIGHT OUTER JOIN**
- Полным **FULL OUTER JOIN**

Ключевое слово **OUTER** можно опустить.

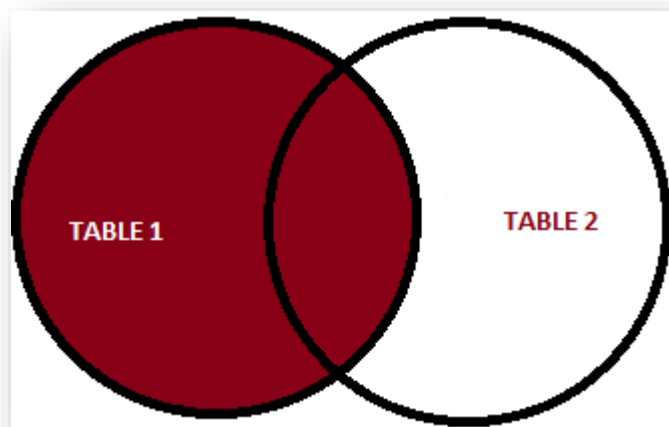


Рисунок 7 - LEFT JOIN

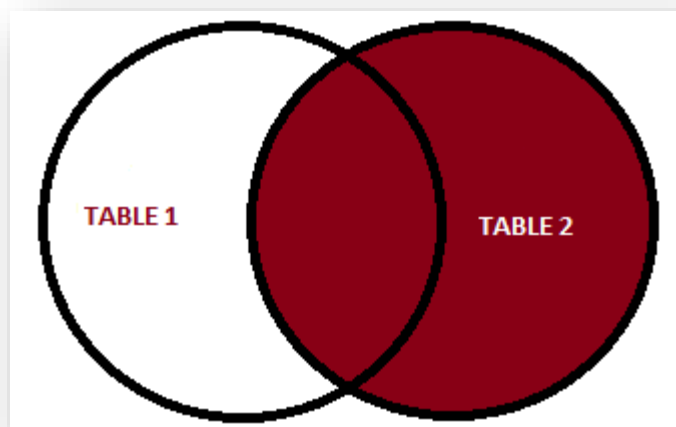


Рисунок 8 - RIGHT JOIN

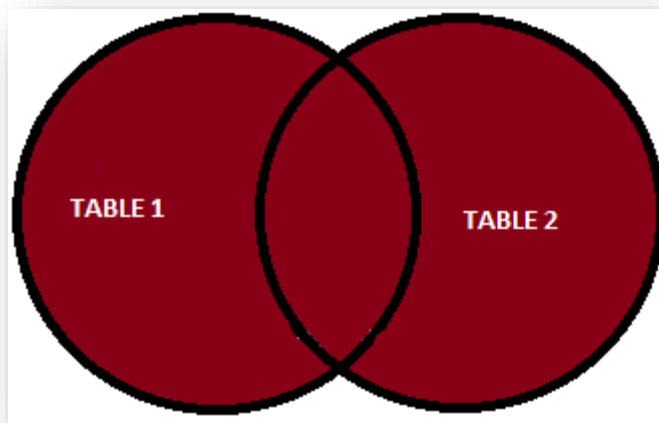


Рисунок 9 - FULL JOIN

Пример 4. Соединим с помощью левого внешнего соединения LEFT OUTER JOIN две таблицы.

```
SELECT NAME, CITYNAME FROM PERSON  
LEFT OUTER JOIN CITY ON PERSON.CID = CITY.CID
```

Результат выполнения запроса:

	NAME	CITYNAME
1	Андрей	Москва
2	Леонид	Санкт-Петербург
3	Сергей	Москва
4	Григорий	NULL

Рисунок 10 – Результат выполнения запроса

Данное внешнее соединение называется левым потому, что оно возвращает все строки из таблицы с левой стороны оператора сравнения, независимо от того, имеются ли совпадающие строки в таблице с правой стороны. Иными словами, данное внешнее соединение возвратит строку с левой таблицы, даже если для нее нет совпадения в правой таблице, со значением NULL соответствующего столбца для всех строк с несовпадающим значением столбца другой, правой, таблицы.

Пример 5. Приведем другой пример из БД библиотеки. Необходимо найти все экземпляры книг, которые никогда не выдавались читателям.

Часть схемы БД с необходимыми таблицами выглядит следующим образом:

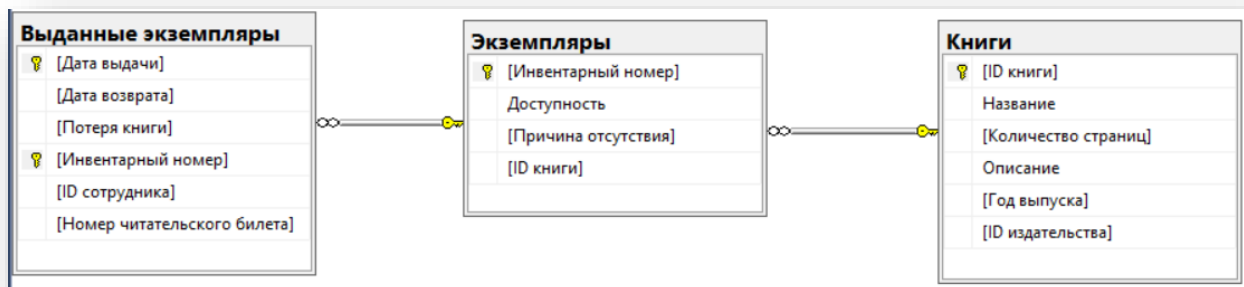


Рисунок 11 – Схема базы данных

Запрос для получения информации:

```
SELECT Экземпляры.[Инвентарный номер], Книги.Название,
Экземпляры.Доступность, Экземпляры.[Причина отсутствия], [Выданные
экземпляры].*
```

```
FROM Экземпляры
```

```
LEFT JOIN [Выданные экземпляры] ON Экземпляры.[Инвентарный номер]
= [Выданные экземпляры].[Инвентарный номер]
```

```
INNER JOIN Книги ON Экземпляры.[ID книги] = Книги.[ID книги]
```

```
WHERE [Выданные экземпляры].[ID сотрудника] IS NULL
```

Далее получим следующий результат:

	Инвентарный номер	Название	Доступность	Причина отсутствия	Дата выдачи	Дата возврата
1	21342148	Гарри Поттер и Дары смерти	1	NULL	NULL	NULL
2	32556459	Гарри Поттер и Дары смерти	1	NULL	NULL	NULL
3	67320984	Гарри Поттер и Дары смерти	1	NULL	NULL	NULL
4	82573485	Гарри Поттер и Дары смерти	0	Продан	NULL	NULL
5	23154679	И вспыхнет пламя	1	NULL	NULL	NULL
6	79832023	И вспыхнет пламя	1	NULL	NULL	NULL
7	12357834	Гарри Поттер и Тайная комната	1	NULL	NULL	NULL
8	23589123	Гарри Поттер и Тайная комната	1	NULL	NULL	NULL
9	23426742	Мастер и Маргарита	0	Потерян	NULL	NULL
10	32587353	Эрагон	0	Передан другой библиотеке	NULL	NULL
11	23578503	Голодные игры	1	NULL	NULL	NULL
12	46379128	Сойка-пересмешница	1	NULL	NULL	NULL
13	23547583	Над пропастью во ржи	0	Потерян	NULL	NULL

Рисунок 12 – Результат запроса

Пример 6. Соединим с помощью правого внешнего соединения RIGHT OUTER JOIN две таблицы.

```
SELECT NAME, CITYNAME FROM PERSON
```

```
RIGHT OUTER JOIN CITY ON PERSON.CID = CITY.CID
```

Результат выполнения запроса:

	NAME	CITYNAME
1	Андрей	Москва
2	Сергей	Москва
3	Леонид	Санкт-Петербург
4	NULL	Казань

Рисунок 13 – Результат выполнения запроса

Операция правого внешнего соединения аналогична левому, но возвращаются все строки таблицы с правой части выражения.

Как можно видеть в результате выполнения запроса, когда для строки из левой таблицы нет совпадающей строки в правой таблице, операция левого внешнего соединения все равно возвращает эту строку, заполняя значением NULL все ячейки соответствующего столбца для несовпадающего значения столбца правой таблицы.

Пример 7. Соединим с помощью полного внешнего соединения FULL OUTER JOIN две таблицы.

```
SELECT NAME, CITYNAME FROM PERSON  
FULL OUTER JOIN CITY ON PERSON.CID = CITY.CID
```

Результат выполнения запроса:

	NAME	CITYNAME
1	Андрей	Москва
2	Леонид	Санкт-Петербург
3	Сергей	Москва
4	Григорий	NULL
5	NULL	Казань

Рисунок 14 – Результат выполнения запроса

Оператор полного внешнего соединения FULL OUTER JOIN соединяет две таблицы. Порядок таблиц для оператора неважен, поскольку оператор является симметричным. Заголовок таблицы-результата является объединением заголовков соединяемых таблиц. Тело результата логически формируется следующим образом. Пусть выполняется

соединение первой и второй таблиц по предикату (условию) *p*. Слова «первой» и «второй» здесь не обозначают порядок в записи (который неважен), а используются лишь для различения таблиц. В результат включается внутреннее соединение (INNER JOIN) первой и второй таблиц по предикату *p*. В результат добавляются те записи первой таблицы, которые не вошли во внутреннее соединение на первом шаге. Для таких записей поля, соответствующие второй таблице, заполняются значениями NULL. В результат добавляются те записи второй таблицы, которые не вошли во внутреннее соединение на первом шаге. Для таких записей поля, соответствующие первой таблице, заполняются значениями NULL.

Пример 8. Приведем пример из БД библиотеки. Необходимо найти все книги, у которых не указано издательство, и все издательства, книги которых не представлены в БД библиотеки.

Часть схемы БД с необходимыми таблицами выглядит следующим образом:

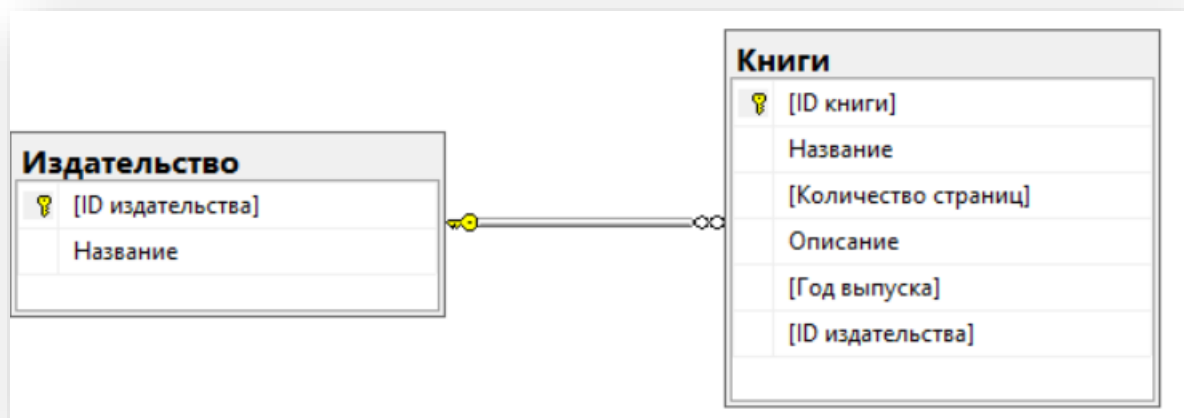


Рисунок 15 – Схема базы данных

Запрос для получения информации:

```
SELECT Книги.Название AS Книга, Издательство.Название AS
```

```
Издательство
```

```
FROM Книги
```

```
FULL JOIN Издательство ON Книги.[ID издательства] =
```

```
Издательство.[ID издательства]
```

```
WHERE Книги.Название IS NULL OR Издательство.Название IS NULL
```

Далее получим следующий результат:

	Книга	Издательство
1	Священная книга оборотня	NULL
2	Чапаев и Пустота	NULL
3	NULL	Клевер Медиа Групп
4	NULL	Москва
5	NULL	Белая ворона

Рисунок 16 – Результат запроса

1.4. Оператор перекрестного соединения (Декартово произведение) CROSS JOIN

Оператор перекрёстного соединения, или декартова произведения CROSS JOIN соединяет две таблицы. Порядок таблиц для оператора неважен, поскольку оператор является симметричным. Заголовок таблицы-результата является объединением (конкатенацией) заголовков соединяемых таблиц. Тело результата логически формируется следующим образом. Каждая строка одной таблицы соединяется с каждой строкой второй таблицы, давая тем самым в результате все возможные сочетания строк двух таблиц.

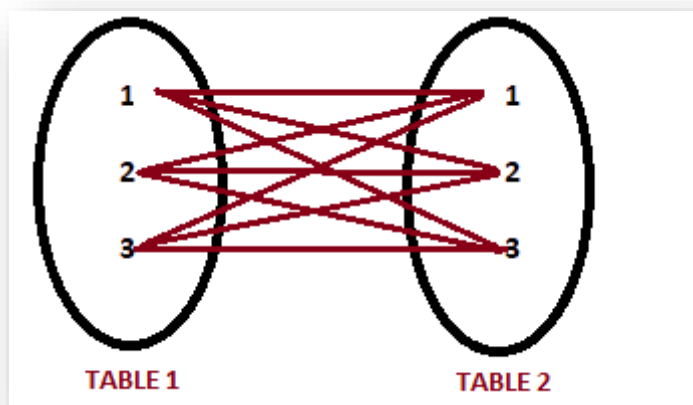


Рисунок 17 - CROSS JOIN

В реальных сценариях операция CROSS JOIN может быть очень полезна при создании отчетов. Например, можно сгенерировать набор дат (например, дни в месяце) (days) и выполнить перекрестное объединение со всеми отделами (departments), имеющимися в базе данных. В результате получится полная таблица день/отдел.

day	department
Jan 01	Dept 1
Jan 01	Dept 2
Jan 01	Dept 3
Jan 02	Dept 1
Jan 02	Dept 2
Jan 02	Dept 3
...	...
Jan 31	Dept 1
Jan 31	Dept 2
Jan 31	Dept 3

Рисунок 28 – Результат CROSS JOIN

Теперь для каждой комбинации день/отдел можно вычислить дневную выручку для данного отдела или другие аналогичные показатели.

Пример 9. Соединим две таблицы при помощи оператора перекрёстного соединения.

```
SELECT * FROM PERSON CROSS JOIN CITY
```

Результат выполнения запроса:

	ID	NAME	CID	CID	CITYNAME
1	1	Андрей	1	1	Москва
2	2	Леонид	2	1	Москва
3	3	Сергей	1	1	Москва
4	4	Григорий	4	1	Москва
5	1	Андрей	1	2	Санкт-Петербург
6	2	Леонид	2	2	Санкт-Петербург
7	3	Сергей	1	2	Санкт-Петербург
8	4	Григорий	4	2	Санкт-Петербург
9	1	Андрей	1	3	Казань
10	2	Леонид	2	3	Казань
11	3	Сергей	1	3	Казань

Рисунок 19 – Результат выполнения запроса

На практике декартово произведение применяется крайне редко. Иногда пользователи получают декартово произведение двух таблиц, когда они забывают включить условие соединения в предложении WHERE при использовании неявного синтаксиса соединения

"старого стиля". В таком случае полученный результат не соответствует ожидаемому, т.к. содержит лишние строки. Наличие неожиданно большого количества строк в результате служит признаком того, что вместо требуемого естественного соединения двух таблиц было получено декартово произведение.

Существует также способ соединения через WHERE. В этом случае таблицы указываются через запятую в секции FROM, а связь находится в секции WHERE. Следующий запрос выводит такой же результат, что и в примере 1:

```
SELECT CITY.*, PERSON.*  
FROM CITY, PERSON  
WHERE PERSON.CID = CITY.CID;
```

Данный способ считается устаревшим и не удобным, поэтому рекомендуется использовать соединение с помощью JOIN.

2. ПРОЦЕДУРЫ

Как правило, в работе с БД используются одни и те же запросы, либо набор последовательных запросов. Хранимые процедуры позволяют объединить последовательность запросов и сохранить их на сервере. Это очень удобный инструмент. По сути хранимые процедуры представляет набор инструкций, которые выполняются как единое целое. Тем самым хранимые процедуры позволяют упростить комплексные операции и вынести их в единый объект.

Также хранимые процедуры позволяют ограничить доступ к данным в таблицах и тем самым уменьшить вероятность преднамеренных или неосознанных нежелательных действий в отношении этих данных.

И еще один важный аспект - производительность. Хранимые процедуры обычно выполняются быстрее, чем обычные SQL-инструкции. Все потому что код процедур компилируется один раз при первом ее запуске, а затем сохраняется в скомпилированной форме.

Таким образом, хранимая процедура имеет три ключевых особенности: упрощение кода, безопасность и производительность.

Для создания хранимой процедуры применяется команда CREATE PROCEDURE или CREATE PROC.

Создадим хранимую процедуру для извлечения таких же данных как в примере 1:

```
USE TEST_JOIN;  
GO  
CREATE PROCEDURE CityPerson AS  
SELECT CITY.*, PERSON.*  
FROM CITY INNER JOIN PERSON  
ON PERSON.CID = CITY.CID;
```


Поскольку команда CREATE PROCEDURE должна вызываться в отдельном пакете, то после команды USE, которая устанавливает текущую базу данных, используется команда GO для определения нового пакета. Для отделения тела процедуры от остальной части скрипта код процедуры нередко помещается в блок BEGIN...END.

После добавления процедуры ее можно увидеть в узле базы данных в SQL Server Management Studio в подузле *Программирование -> Хранимые процедуры*.

Для выполнения хранимой процедуры вызывается команда EXEC или EXECUTE:

```
EXEC CityPerson
```

Для удаления процедуры применяется команда DROP PROCEDURE:

```
DROP PROCEDURE CityPerson
```

Для модификации структуры хранимых процедур используется инструкция ALTER PROCEDURE (или ALTER PROC).

Инструкция ALTER PROCEDURE обычно применяется для изменения инструкций Transact-SQL внутри процедуры. Все параметры инструкции ALTER PROCEDURE имеют такое же значение, как и одноименные параметры инструкции CREATE PROCEDURE.

Процедуры могут принимать параметры. Параметры бывают входными - с их помощью в процедуру можно передать некоторые значения. И также параметры бывают выходными - они позволяют вернуть из процедуры некоторое значение.

Например, определим процедуру для БД библиотеки, которая будет возвращать количество книг, написанных одним конкретным автором.

```
CREATE PROC [Количество книг]
@authorsurname NVARCHAR(50),
@authorname NVARCHAR(50),
@authorpatronymic NVARCHAR(50) = NULL
AS
SELECT Авторы.Фамилия, Авторы.Имя, Авторы.Отчество, COUNT (*) AS
[Количество книг]
FROM Авторы INNER JOIN
[Авторы книг] ON Авторы.[ID автора] = [Авторы книг].[ID
автора] INNER JOIN
Книги ON [Авторы книг].[ID книги] = Книги.[ID книги]
WHERE Авторы.Фамилия = @authorsurname AND Авторы.Имя = @authorname
AND Авторы.Отчество = @authorpatronymic
GROUP BY Авторы.Фамилия, Авторы.Имя, Авторы.Отчество
```

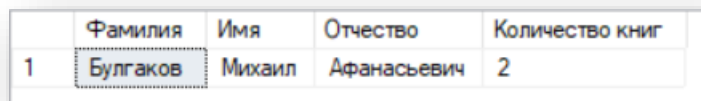
После названия процедуры идет список входных параметров, которые определяются также, как и переменные - название начинается с символа @, а после названия идет тип переменной. При вызове процедуры ей через запятую передаются значения параметров.

При этом значения передаются параметрам процедуры в том же порядке, в котором объявлены сами параметры.

Параметры также можно отмечать как необязательные, присваивая им некоторое значение по умолчанию. Например, в данной ХП для параметра @authorpatronymic автоматически устанавливается значение NULL, если соответствующее значение не передано в процедуру.

Пример выполнения ХП и результат:

EXEC [Количество книг] 'Булгаков', 'Михаил', 'Афанасьевич'



	Фамилия	Имя	Отчество	Количество книг
1	Булгаков	Михаил	Афанасьевич	2

Рисунок 19 – Результат выполнения ХП

3. ФУНКЦИИ

Команда CREATE FUNCTION создаёт определяемую пользователем функцию в SQL Server. Определяемая пользователем функция представляет собой подпрограмму Transact-SQL, которая принимает параметры, выполняет действия, такие как сложные вычисления, а затем возвращает результат этих действий в виде значения. Возвращаемое значение может быть скалярным значением или таблицей. При помощи этой инструкции можно создать подпрограмму, которую можно повторно использовать следующими способами.

- В инструкциях Transact-SQL, например, SELECT.
- В приложениях, вызывающих функцию.
- В определении другой пользовательской функции.
- Для параметризации представления или улучшения функциональности индексированного представления.
- Для определения столбца таблицы.
- Для определения ограничения CHECK на столбец.
- Для замены хранимой процедуры.

Для примера создадим функцию, которая будет возвращать книги из БД библиотеки, которые прочитаны определенное количество раз.

```
CREATE FUNCTION dbo.read_books (@num int)
RETURNS TABLE
AS
RETURN
(
    SELECT Книги.Название AS [Название книги], COUNT (*) AS
[Количество прочитанных]
    FROM [Выданные экземпляры]
    INNER JOIN Экземпляры ON [Выданные экземпляры].[Инвентарный
номер] = Экземпляры.[Инвентарный номер]
    AND [Выданные экземпляры].[Дата возврата] IS NOT NULL
    INNER JOIN Книги ON Экземпляры.[ID книги] = Книги.[ID книги]
    GROUP BY Книги.Название
    HAVING COUNT (*) = @num
);
```

После CREATE FUNCTION указывается имя схемы, к которой принадлежит определяемая пользователем функция. Далее через точку указывается имя определяемой пользователем функции. Имена функций должны удовлетворять правилам построения идентификаторов и должны быть уникальными в пределах базы данных и схемы.

В скобках находится параметр пользовательской функции. Может быть объявлено один или несколько параметров. Для функций допускается не более 2100 параметров. При выполнении функции значение каждого из объявленных параметров должно быть указано пользователем, если для них не определены значения по умолчанию.

Для определения имени параметра используется знак @ как первый символ. Параметры являются локальными в пределах функции, в разных функциях могут быть использованы одинаковые имена параметров. Параметры могут использоваться только вместо констант. Они не могут использоваться вместо имен таблиц, имен столбцов или имен других объектов базы данных. Для каждого параметра также указывается тип данных. Для функций Transact-SQL допустимы любые типы данных за исключением timestamp.

TABLE указывает, что возвращаемым значением функции с табличным значением, является таблица. Функции могут также возвращать скалярное значение. Для этого необходимо указать тип данных возвращаемого значения.

При вызове этой функции выполняется следующий запрос.

```
SELECT * FROM dbo.read_books (3);
```

Результаты функции представлены на рисунке 20.

	Название книги	Количество прочитанных
1	Вокруг света за 80 дней	3
2	Собачье сердце	3

Рисунок 20 – Результат работы функции

ЗАКЛЮЧЕНИЕ

Представленный в данном методическом указании материал предоставляет возможность получить базовые теоретические знания о операторах соединения в языке SQL. Примеры и задания дают возможность получить практический опыт по составлению запросов для выборки с различными формами соединений.