Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа «Компьютерных технологий и информационных систем»

**ОТЧЕТ по лабораторной работе №3**

по дисциплине «Базы данных»

**Выполнил:**

студент группы 5130902/20201 \_\_\_\_\_\_\_\_\_\_\_\_\_ А. И. Сафонов

подпись

**Проверил:**

Кандитад тех. Наук., доцент \_\_\_\_\_\_\_\_\_\_\_\_ С. А. Нестеров

подпись

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_ 2025г.

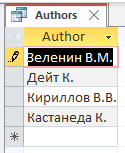
Санкт-Петербург, 2025

Оглавление

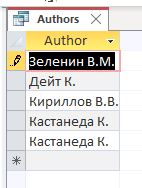
[СОЗДАНИЕ ЗАПРОСОВ В КОНСТРУКТОРЕ 3](#_Toc190192782)

Данная лабораторная работа, как и предыдущая, посвящена созданию запросов. Мы рассмотрим еще один режим работы с запросами – режим SQL.

***Задание***. Используя базу lib.accdb из второй лабораторной работы, в конструкторе создайте запрос, выводящий список авторов книг из таблицы Book (нам нужны только авторы, без названий книг и т.д.).



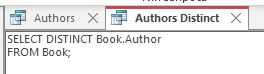
Добавьте в таблицу данные таким образом, чтобы в ней было несколько книг одного автора. Выполните запрос и обратите внимание на то, что если автор написал несколько книжек, то в результатах запроса он будет фигурировать тоже несколько раз.

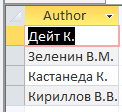


Перейдите в режим SQL, в котором Вы увидите то предложение SQL, которое и является запросом: SELECT Book.Author FROM Book;



В SQL для указания на то, что нужно отбрасывать повторения, служит ключевое слово DISTINCT. Измените запрос так, как показано ниже и проверьте его работу. SELECT DISTINCT Book.Author FROM Book;

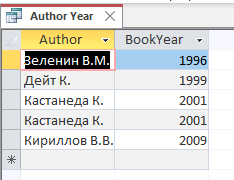


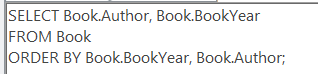


Нужно отметить, что в выражении SELECT можно явно указать, что надо возвращать все записи, включая дубликаты. Для этого используется ключевое слово ALL, но обычно его не ставят, т.к. оно подразумевается, если ничего другого явно не указано.

Если требуется задать определенный порядок сортировки выводимых записей, в режиме SQL это можно сделать, добавив инструкцию ORDER BY, после указывается столбец или столбцы, по которым проводится сортировка. По умолчанию или при явном добавлении ключевого слова ASC сортировка проводится по возрастанию, если стоит DESC – по убыванию.

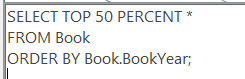
***Задание.*** Напишите запрос, выводящий информацию о книгах из таблицы Book в порядке от самых старых к самым новым. Книги с одинаковым годом издания упорядочивайте в списке по названию в алфавитном порядке.

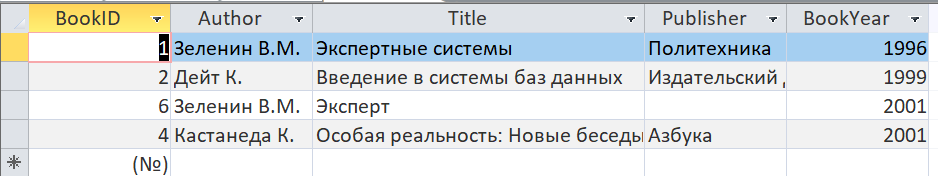




В некоторых случаях нужно вывести не весь сортированный список, а какую-то его часть. Например, нас могут интересовать первые 5 записей из списка или первые 10 процентов списка. В подобных случаях, в запрос SELECT добавляют соответственно «TOP n» или «TOP n PERCENT», где n – количество записей или процент. Например, следующий запрос выведет список из двух самых новых изданий из Book (символ \* указывает на то, что нам нужны все столбцы).

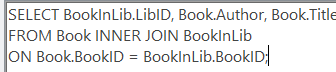
***Задание.*** Напишите запрос, выводящий информацию о книгах, составляющих первую половину общего списка из таблицы Book, упорядоченного по возрастанию года (т.е. самые старые книги)

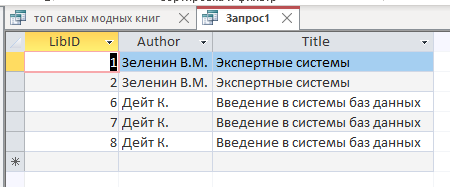




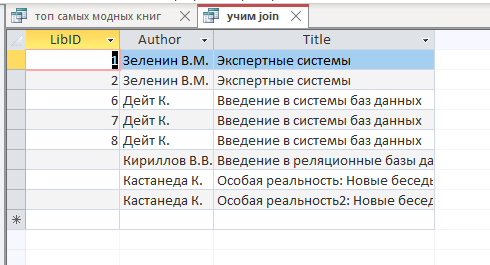
Инструкция INNER JOIN указывает на то, что надо соединить таблицы по условию совпадения значений в столбцах Book.BookID и BookInLib.BookID. Причем соединение должно быть «внутренним» – соответствующие значения должны встречаться как в одной, так и в другой таблице. Кроме внутреннего, Access также поддерживает левое и правое внешнее соединение – LEFT JOIN и RIGHT JOIN. В первом случае в выборку берутся все записи из левой таблицы и соответствующие им записи из правой, если они есть. Во втором случае – наоборот.

***Задание.*** Скопируйте приведенный выше запрос в свою базу. Выполните его.



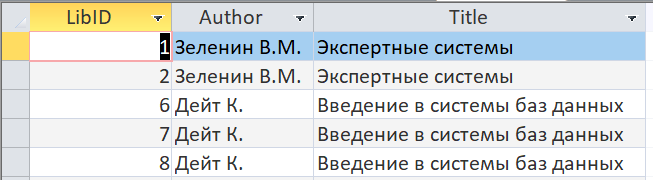


При необходимости измените набор данных в таблицах так, чтобы в таблице Book были записи, не связанные с записями из таблицы BookInLib. В запросе замените INNER JOIN на LEFT JOIN, проанализируйте результат.



Выборкой берутся все записи из левой таблицы (Book) и соответствующие им записи из правой (BookInLib).

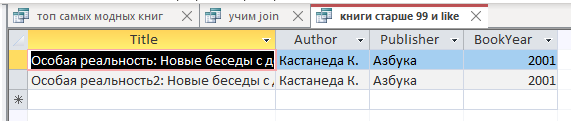
Объясните, почему для нашей структуры базы данных замена в этом запросе INNER JOIN на RIGHT JOIN не приведет к изменениям в выводимом наборе данных (тут надо вспомнить о внешних ключах, которые создавались в ходе выполнения первой лабораторной).

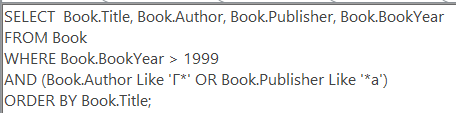


Выборкой берутся все записи из правой (BookInLib) таблицы и соответствующие им записи из левой (Book). Но так как BookID в BookInLib — это внешний ключ на Book.BookID, то в BookInLib не может быть записей с BookID, которых нет в Book. Это значит, что каждая строка в BookInLib уже имеет соответствующую запись в Book, поэтому результат не изменится.

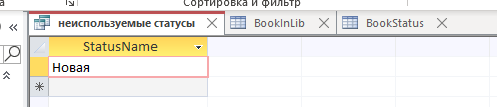
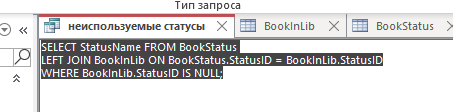
Как мы разбирали в предыдущей лабораторной работе, условия могут включать проверку на равенство, неравенство, принадлежность множеству (IN), соответствие текстовой маске (LIKE) и т.д. Условие также может быть составным и включать логические операторы NOT (отрицание), AND (и), OR (или); их приоритет убывает именно в такой последовательности.

***Задание.*** Напишите на SQL запрос, который выведет в обратном алфавитном порядке названия книг, выпущенных после 1999 года, фамилия автора (значение поля Author) у которых начинается с буквы «Г» или название издательства заканчивается буквой «а».

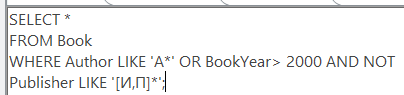


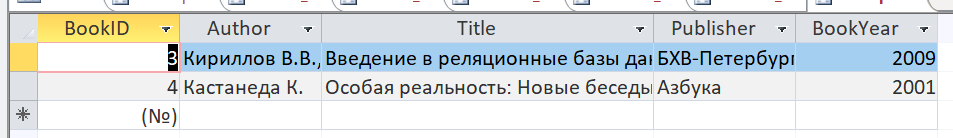


***Задание.*** Напишите на SQL запрос, возвращающий список названий неиспользуемых статусов книг, то есть таких, что данный статус не назначен ни одной книге в таблице BookInLib. Подсказка: нужно использовать внешнее соединение таблиц и условие IS NULL. При необходимости отредактируйте используемый набор данных, чтобы такие статусы были, и проверьте работу запроса.

***Задание.*** Опишите, что на ваш взгляд делает приведённый ниже запрос. Скопируйте его код в Access и выполните. Правильным ли оказалось ваше предположение?





Запрос ищет книги, у которых автор начинается на "A" ИЛИ  
они изданы после 2000 года и издательство не начинается с "И" или "П".

***Реляционная модель данных***

Реляционная модель данных — это подход к организации и управлению данными, основанный на математической теории множеств и логике первого порядка. Она была предложена Эдгардом Коддом в 1970 году и стала основой для современных систем управления базами данных (СУБД).

Реляционная модель данных описывает:

Структуры данных в виде (изменяющихся во времени) наборов отношений.

Теоретико-множественные операции над данными: объединение, пересечение, разность и декартово произведение.

Специальные реляционные операции: селекция, проекция, соединение и деление.

Специальные правила, обеспечивающие целостность данных.

Основными понятиями реляционных баз данных являются:

тип данных,

домен,

атрибут,

кортеж,

первичный ключ

и отношение.

***Тип данных***

Понятие **типа данных** в реляционной модели данных во многом аналогично понятию типа данных в языках программирования. Типы данных в реляционных базах данных определяют, какие значения могут храниться в атрибутах (столбцах) таблиц, и обеспечивают корректность операций с данными.

Типы данных в реляционных базах данных можно разделить на несколько категорий:

**Символьные данные (строки)**:

CHAR(n) — строка фиксированной длины (n символов).

VARCHAR(n) — строка переменной длины (до n символов).

TEXT — строка произвольной длины.

**Числовые данные**:

INTEGER (или INT) — целое число.

SMALLINT — малое целое число.

BIGINT — большое целое число.

DECIMAL(p, s) или NUMERIC(p, s) — точные числа с фиксированной точностью (p — общее количество цифр, s — количество цифр после запятой).

FLOAT или REAL — числа с плавающей точкой.

**Специализированные числовые данные**:

MONEY — тип для хранения денежных значений (обычно с фиксированной точностью).

**Темпоральные данные (временные)**:

DATE — дата (год, месяц, день).

TIME — время (часы, минуты, секунды).

TIMESTAMP — дата и время.

INTERVAL — временной интервал.

**Другие типы**:

BOOLEAN — логический тип (TRUE/FALSE).

BLOB (Binary Large Object) — для хранения бинарных данных (например, изображений).

JSON или XML — для хранения структурированных данных в форматах JSON или XML.

Типы данных в реляционных БД действительно похожи на типы данных в языках программирования, но есть и отличия:

В БД типы данных часто имеют более строгие ограничения (например, фиксированная длина строки CHAR(n)).

В БД типы данных могут быть оптимизированы для хранения и обработки больших объемов данных.

В БД поддерживаются специализированные типы (например, DATE, MONEY), которые редко встречаются в языках программирования в явном виде.

***Домен***

Понятие домена более специфично для баз данных, хотя и имеет некоторые аналогии с подтипами в некоторых языках программирования. Оно играет ключевую роль в обеспечении семантической целостности данных и их корректной интерпретации.

**Домен** — это множество допустимых значений для атрибута (столбца) таблицы, которое определяется на основе: 1. **Базового типа данных** (например, целые числа, строки, даты). 2. **Ограничений (логических выражений)**, которые задают дополнительные правила для значений.

Домен можно рассматривать как **пользовательский тип данных**, который уточняет базовый тип, добавляя к нему семантические и логические ограничения.

Домен несет в себе **семантическую нагрузку**, то есть определяет смысл данных.

Преимущества использования доменов: 1. **Семантическая ясность**: домены делают данные более понятными, так как они отражают их смысл. 2. **Целостность данных**: ограничения домена предотвращают попадание некорректных значений в базу данных. 3. **Упрощение проектирования**: домены позволяют повторно использовать типы данных с одинаковыми ограничениями. 4. **Сравнимость данных**: данные считаются сравнимыми только в пределах одного домена, что предотвращает логические ошибки.

***Атрибут***

Понятие **атрибута** в базах данных связано с полями записей и их характеристиками.

**Атрибут** — это свойство или характеристика сущности (объекта), которая описывается в таблице базы данных. В реляционной модели данных атрибут соответствует **столбцу таблицы** и характеризует:

**Имя атрибута** — уникальное название столбца (например, FirstName, Age, Salary).

**Тип данных или домен** — определяет, какие значения могут храниться в атрибуте (например, целые числа, строки, даты).

***Запись***

Понятие записи в реляционных базах данных связано со строками таблицы, которые содержат данные в виде набора полей.

**Запись** (или **кортеж**) — это строка в таблице базы данных, которая представляет собой набор значений, соответствующих атрибутам (столбцам) таблицы. Каждая запись описывает один экземпляр сущности (объекта), например, одного сотрудника, один заказ или один товар.

Запись состоит из **полей**, каждое из которых соответствует определенному атрибуту таблицы. Поля могут содержать данные разных типов, таких как:

Целые числа (INT).

Строки (VARCHAR, TEXT).

Даты (DATE).

Числа с плавающей точкой (DECIMAL, FLOAT).

Каждая запись в таблице должна быть уникальной. Это обеспечивается с помощью **первичного ключа** (Primary Key). Поля в записи соответствуют порядку атрибутов в таблице.

***Первичный ключ***

Понятие **ключа отношения** (или просто **ключа**) в реляционных базах данных является фундаментальным для обеспечения уникальности и целостности данных.

**Ключ отношения** — это атрибут (или набор атрибутов) таблицы, который **однозначно идентифицирует каждый кортеж (запись)** в этой таблице. Ключ гарантирует, что никакие две записи в таблице не могут иметь одинаковые значения для этого атрибута (или набора атрибутов).

Бывают ключи двух типов.

**Простой ключ**:

Состоит из одного атрибута.

Пример: В отношении **СОТРУДНИКИ** атрибут СОТР\_НОМЕР может быть простым ключом, если каждый сотрудник имеет уникальный (например, табельный) номер.

**Составной ключ**:

Состоит из нескольких атрибутов.

Пример: Номер паспорта может состоять из атрибутов СЕРИЯ и НОМЕР.

*Реляционная алгебра*

Отношения являются основой для операций **реляционной алгебры**, которые используются для манипуляции данными в реляционных базах данных. Основные операции включают:

**Выборка (SELECT)** — извлечение строк, удовлетворяющих условию.

**Проекция (PROJECT)** — извлечение определенных столбцов.

**Объединение (JOIN)** — комбинирование данных из нескольких отношений.

**Декартово произведение (CARTESIAN PRODUCT)** — создание всех возможных комбинаций строк из двух отношений.

*Фундаментальные свойства отношений*

Рассмотрим некоторые важные свойства отношений:

**Отсутствие кортежей-дубликатов**. То свойство, что отношения не содержат кортежей-дубликатов, следует из определения отношения как множества кортежей. В классической теории множеств по определению каждое множество состоит из различных элементов. Однако при формальном определении первичного ключа требуется обеспечение его «минимальности», т.е. в набор атрибутов первичного ключа не должны входить такие атрибуты, которые можно отбросить без ущерба для основного свойства - однозначно определять кортеж.

**Отсутствие упорядоченности кортежей**. Свойство отсутствия упорядоченности кортежей отношения также является следствием определения отношения-экземпляра как множества кортежей. Отсутствие требования к поддержанию порядка на множестве кортежей отношения дает дополнительную гибкость СУБД при хранении баз данных во внешней памяти и при выполнении запросов к базе данных.

**Отсутствие упорядоченности атрибутов**. Атрибуты отношений не упорядочены. Для ссылки на значение атрибута в кортеже отношения всегда используется имя атрибута. Это свойство теоретически позволяет, например, модифицировать схемы существующих отношений не только путем добавления новых атрибутов, но и путем удаления существующих атрибутов.

**Атомарность значений атрибутов**. Значения всех атрибутов являются атомарными. Это следует из определения домена как потенциального множества значений простого типа данных, т.е. среди значений домена не могут содержаться множества значений (отношения). (Атомарность – операция, которая выполняется либо целиком, либо вовсе не выполняется. Например, в системе бронирования обе задачи (бронирование мест и обновление сведений о клиенте) должны выполняться как единая транзакция.)

*Целостность сущности и ссылок*

Наконец, в целостной части реляционной модели данных фиксируются два базовых требования целостности, которые должны поддерживаться в любой реляционной СУБД. Первое требование называется требованием **целостности сущностей**. Объекту или сущности реального мира в реляционных БД соответствуют кортежи отношений. Конкретно требование состоит в том, что любой кортеж любого отношения отличим от любого другого кортежа этого отношения, т.е., другими словами, любое отношение должно обладать первичным ключом. Как мы видели в предыдущем разделе, это требование автоматически удовлетворяется, если в системе не нарушаются базовые свойства отношений.

Второе требование называется требованием **целостности по ссылкам** и является несколько более сложным. Очевидно, что при соблюдении нормализованности отношений сложные сущности реального мира представляются в реляционной БД в виде нескольких кортежей нескольких отношений.

**Требование целостности по ссылкам**, или требование внешнего ключа состоит в том, что для каждого значения внешнего ключа, появляющегося в ссылающемся отношении, в отношении, на которое ведет ссылка, должен найтись кортеж с таким же значением первичного ключа, либо значение внешнего ключа должно быть неопределенным (т.е. ни на что не указывать). Для нашего примера это означает, что если для сотрудника указан номер отдела, то этот отдел должен существовать.

Понятно, что при **обновлении** ссылающегося отношения (вставке новых кортежей или модификации значения внешнего ключа в существующих кортежах) достаточно следить за тем, чтобы не появлялись некорректные значения внешнего ключа. Но как быть при **удалении** кортежа из отношения, на которое ведет ссылка? Здесь существуют три подхода, каждый из которых поддерживает целостность по ссылкам. Первый подход заключается в том, что запрещается производить удаление кортежа, на который существуют ссылки (т.е. сначала нужно либо удалить ссылающиеся кортежи, либо соответствующим образом изменить значения их внешнего ключа). При втором подходе при удалении кортежа, на который имеются ссылки, во всех ссылающихся кортежах значение внешнего ключа автоматически становится неопределенным. Наконец, третий подход (каскадное удаление) состоит в том, что при удалении кортежа из отношения, на которое ведет ссылка, из ссылающегося отношения автоматически удаляются все ссылающиеся кортежи.

Преимущества реляционного подхода

Преимущества реляционного подхода достаточно очевидны:

Предсказуемость результатов работы с данными. В основе реляционной модели лежит математическая модель, следовательно, любой запрос к базе данных, составленный на корректном языке влечет ответ, однозначно определяемый схемой БД и конкретными данными. При этом пользователю не требуется информация о физической организации данных.

Предметная область часто достаточно естественно описывается в терминах таблиц.

Основное (подразумеваемое) правило:

Система, которая провозглашается поставщиком как реляционная СУБД, должна управлять базами данных исключительно способами, соответствующими реляционной модели.

Конкретные требования к реляционной СУБД раскрываются в следующих правилах:

**Информационное правило**. Вся информация, хранимая в базе данных, должна быть представлена единственным образом: в виде значений в реляционных таблицах.

**Правило гарантированного логического доступа**. К каждому имеющемуся в реляционной базе атомарному значению должен быть гарантирован доступ с помощью указания имени таблицы, значения первичного ключа и имени атрибута.

**Правило наличия значения** (missing information). В полностью реляционной СУБД должны иметься специальные индикаторы (отличные от пустой символьной строки или строки из одних пробелов и отличные от нуля или какого-то другого числового значения) для выражения (на логическом уровне, не зависимо от типа данных) того факта, что значение отсутствует по меньшей мере по двум различным причинам: его действительно нет, либо оно не применимо к данной позиции. СУБД должна не только отражать этот факт, но и распространять на такие индикаторы свои функции манипулирования данными не зависимо от типа данных. Как правило это значение обозначается **NULL**.

**Правило динамического реляционного каталога**. Описание базы данных выглядит логически как обычные данные, так что авторизованные пользователи и прикладные программы могут употреблять для работы с этим описанием тот же реляционный язык, что и при работе с обычными данными.

**Правило полноты языка работы с данными**. Сколько бы много в СУБД ни поддерживалось языков и режимов работы с данными, должен иметься по крайней мере один язык, выразимый в виде командных строк в некотором удобном синтаксисе, который бы позволял формулировать:

определение данных;

определение правил целостности;

манипулирование данными (в диалоге и из программы);

определение таблиц-представлений (в том числе и возможности их модификации);

определение правил авторизации;

границы транзакций.

**Правило модификации таблиц-представлений**. В СУБД должен существовать корректный алгоритм, позволяющий автоматически для каждой таблицы-представления определять во время ее создания, может ли она использоваться для вставки и удаления строк и какие из столбцов допускают модификацию, и заносящий полученную таким образом информацию в системный каталог.

**Правило множественности операций**. Возможность оперирования базовыми таблицами или таблицами-представлениями распространяется полностью не только на выдачу информации из БД, но и на вставку, модификацию и удаление данных.

**Правило физической независимости**. Диалоговые операторы и прикладные программы на логическом уровне не должны страдать от каких-либо изменений во внутреннем хранении данных или методах доступа СУБД

**Правило логической независимости**. Диалоговые операторы и прикладные программы на логическом уровне не должны страдать от таких изменений в базовых таблицах, которые сохраняют информацию и теоретически допускают неизменность этих операторов и программ.

**Правило сохранения целостности**. Диалоговые операторы и прикладные программы не должны изменяться при изменении правил целостности в БД, задаваемых языком работы с данными и хранимых в системном каталоге.

**Правило независимости от распределенности**. Диалоговые операторы и прикладные программы на логическом уровне не должны зависеть от совершаемого физического разнесения данных (если первоначально СУБД работала с нераспределенными данными) или перераспределения (если СУБД распределенная).

**Правило ненарушения реляционного языка**. Если в реляционной СУБД имеется язык низкого уровня (для работы с отдельными строками), он не должен позволять нарушать или «обходить» правила, сформулированные на языке высокого уровня (множественном) и занесенные в системный каталог.

Яндекс практикум

