

Министерство образования Республики Беларусь

Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра электронных вычислительных машин

Е.В. Калабухов

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

по дисциплине «Базы данных, знаний и экспертные системы»
для студентов специальности I-40 02 01
«Вычислительные машины, системы и сети»
всех форм обучения

Минск 2007

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ	3
ЛАБОРАТОРНАЯ РАБОТА №1. СОЗДАНИЕ ER-МОДЕЛИ ДАННЫХ	4
ЛАБОРАТОРНАЯ РАБОТА №2. СОЗДАНИЕ РЕЛЯЦИОННОЙ МОДЕЛИ ДАННЫХ ПО ER-МОДЕЛИ	11
ЛАБОРАТОРНАЯ РАБОТА №3. НОРМАЛИЗАЦИЯ РЕЛЯЦИОННОЙ МОДЕЛИ ДАННЫХ МЕТОДОМ ДЕКОМПОЗИЦИИ	33
ЛАБОРАТОРНАЯ РАБОТА №4. РАБОТА С РЕЛЯЦИОННОЙ БД НА ЯЗЫКЕ SQL	42
ЛИТЕРАТУРА	54

ПРЕДИСЛОВИЕ

Цели настоящего практикума – изучение общей методологии проектирования баз данных, а также изучение языка DML SQL, для выполнения запросов к полученной базе данных.

В рамках общей методологии проектирования охватываются следующие основные задачи:

- 1) концептуальное проектирование базы данных на основе построения ER-диаграммы некоторого фрагмента реального мира;
- 2) логическое проектирование базы данных на основе построения предварительной реляционной модели данных на основании ER-диаграммы, выполнение нормализации предварительной реляционной модели данных и реализация полученной модели в среде целевой СУБД MS Access;

Выполнение запросов проводится с использованием языка SQL в среде целевой СУБД MS Access и делится на запросы выборки и запросы модификации данных.

Все работы практикума связаны между собой этапами проектирования и реализации реляционной базы данных, поэтому сдача работы фактически означает допуск к выполнению следующей.

Следует заметить, что использование СУБД MS Access связано не только с изучением данного продукта, как с его распространенностью, простотой работы и возможностью использования за пределами лаборатории. Все лабораторные работы, по желанию студента, можно выполнить в любой другой реляционной СУБД.

ЛАБОРАТОРНАЯ РАБОТА №1

СОЗДАНИЕ ER-МОДЕЛИ ДАННЫХ

1. Цель работы

Познакомиться с ER-моделью данных (моделью «сущность-связь»). Для указанного варианта задания разработать ER-модель данных (с учетом семантических ограничений предметной области). Представить модель в виде ER-диаграммы.

2. Краткие теоретические сведения

Модель данных является представлением «реального мира» (т.е. реальных объектов и событий, и связей между ними), это некоторая абстракция, в которой остаются только те части реального мира, которые важны для разработчиков БД, а все второстепенные – игнорируются.

Цель построения модели данных – представление данных в понятном виде, который легко можно применить при проектировании БД (модель должна точно и недвусмысленно описывать части реального мира в таком виде, который позволяет разработчикам и пользователям (заказчикам) БД обмениваться мнениями при разработке и поддержке БД).

Модель «сущность-связь» (ER-модель) представляет собой высокоуровневую концептуальную модель данных.

2.1. Концепции ER-модели

1) Объекты (типы сущностей).

Типы объектов (типы сущностей) – множество объектов реального мира с

одинаковыми свойствами, характеризуются независимым существованием и могут быть объектом как с реальным (физическим) существованием (например, «работник», «деталь», «поставщик»), так и объектом с абстрактным (концептуальным) существованием (например, «рабочий стаж», «осмотр объекта»). Разные разработчики могут выделять разные типы объектов. Каждый тип объекта идентифицируется именем и списком свойств.

Объект (экземпляр типа объекта или сущность) – экземпляр типа сущности, «предмет, который может быть четко идентифицирован» на основе свойств (т.к. обладает уникальным набором свойств среди объектов одного типа).

Типы объектов классифицируются как сильные и слабые:

- слабый тип объекта (дочерний, зависимый или подчиненный) – тип объекта, существование которого зависит от какого-то другого типа объекта;
- сильный тип объекта (родительский, владелец или доминантный) – тип объекта, существование которого не зависит от какого-то другого типа объекта.

Представление объектов на диаграмме: сильный тип объекта – прямоугольник, с именем внутри него; слабый тип объекта – прямоугольник с двойным контуром (деление объектов на сильные и слабые – по желанию проектировщика).

2) Свойства (атрибуты).

Свойства (атрибуты) – служат для описания типов объектов или отношений. Значения свойств каждого типа извлекаются из соответствующего множества значений (в этом множестве определяются все потенциальные значения свойства, различные свойства могут использовать одно множество значений).

Свойства делят по характеристикам:

- простые и составные: простое свойство состоит из одного компонента с

независимым существованием (такое свойство – атомарное (не может быть разделено на более мелкие компоненты); например «зарплата», «пол»); составное свойство – состоит из нескольких компонентов, каждый из которых характеризуется независимым существованием (могут быть разделены на более мелкие части (например, «адрес»), но решение о представлении такого атрибута (как простой или составной) зависит от проектировщика);

- однозначные и многозначные: однозначное свойство – свойство, которое может содержать только одно значение для одного объекта; многозначное свойство – может содержать несколько значений для одного объекта (например, «телефон компании»);
- производные и базовые: производное свойство - представляет значение, производное от значения связанного с ним свойства или некоторого множества свойств, принадлежащих некоторому типу объектов (не обязательно одному), например, «сумма деталей», «возраст сотрудника»; базовое – не зависит от других свойств.
- ключевое и не ключевое: ключ – свойство (набор свойств), которое однозначно определяет объект из всех объектов данного типа (например, «номер паспорта»); ключи делят на: потенциальные – содержит значения, которые позволяют уникально идентифицировать каждый отдельный экземпляр типа объекта (тип объекта может иметь несколько потенциальных ключей); первичные – один из потенциальных ключей для типа объекта, который выбран для идентификации объектов (должен гарантировать уникальность значений не только в текущий период, но и в обозримом будущем, лучше чтобы он содержал меньшее число свойств (например, «номер сотрудника», «номер паспорта»)); альтернативные – все потенциальные ключи, кроме первичного; составной ключ – потенциальный ключ, который состоит из 2-х и более свойств (позволяет определить уникальность объекта, не вводя лишних уникальных свойств).
Представление свойств на диаграмме: в виде эллипсов с именем внутри

него, присоединенных линией к типу объекта; для производных свойств – эллипс окружен пунктирным контуром, для многозначных – двойным; имя свойства, которое является первичным ключом – подчеркивается.

3) Отношения (типы связей).

Типы отношений (типы связи) – осмысленная ассоциация (связь) между типами объектов, каждое отношение имеет имя, описывающее его функцию.

Экземпляр отношения (отношение) – ассоциация (связь) между экземплярами объектов, включающая по одному экземпляру объекта с каждой стороны связи.

Объекты, включенные в отношение, называются участниками этого отношения (при этом в связях для определения функций каждого участника могут присваиваться ролевые имена). Количество участников данного отношения называется степенью этого отношения (два участника – бинарная (наиболее часто применима), три – тернарная, четыре – кватернарная, n-участников – n-арная). Существуют унарные (рекурсивные) отношения - в них одни и те же типы объектов участвуют несколько раз и в разных ролях.

Представление отношений на диаграмме: в виде ромба с указанным в нем именем связи и соединенного линиями с участниками отношения.

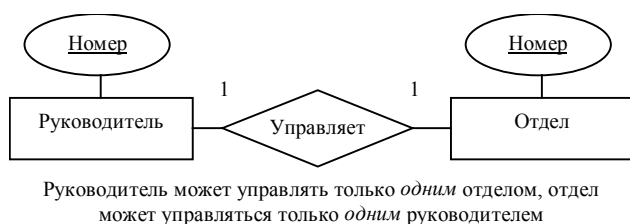
2.2. Структурные ограничения ER-модели

Структурные ограничения, накладываемые на участников отношения, являются отражением требований реального мира.

Показатель кардинальности (мощность отношения) – максимальное количество элементов одного типа объекта, связанных с одним элементом другого типа объекта. Обычно рассматриваются следующие виды связей:

- «один к одному» – максимальная мощность отношения в обоих направлениях равна одному;

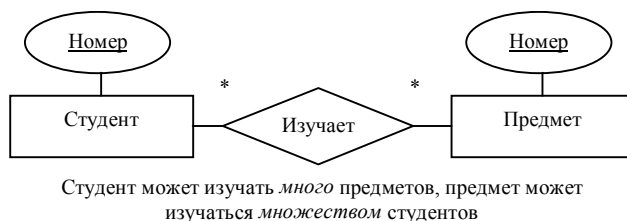
- «один ко многим» - максимальная мощность отношения в одном направлении равна одному, а в другом – многим;
- «многие ко многим» - максимальная мощность отношения в обоих направлениях равна многим.



Отношение «один к одному».



Отношение «один ко многим».



Отношение «многие ко многим».

По степени участия объектов в отношении выделяют:

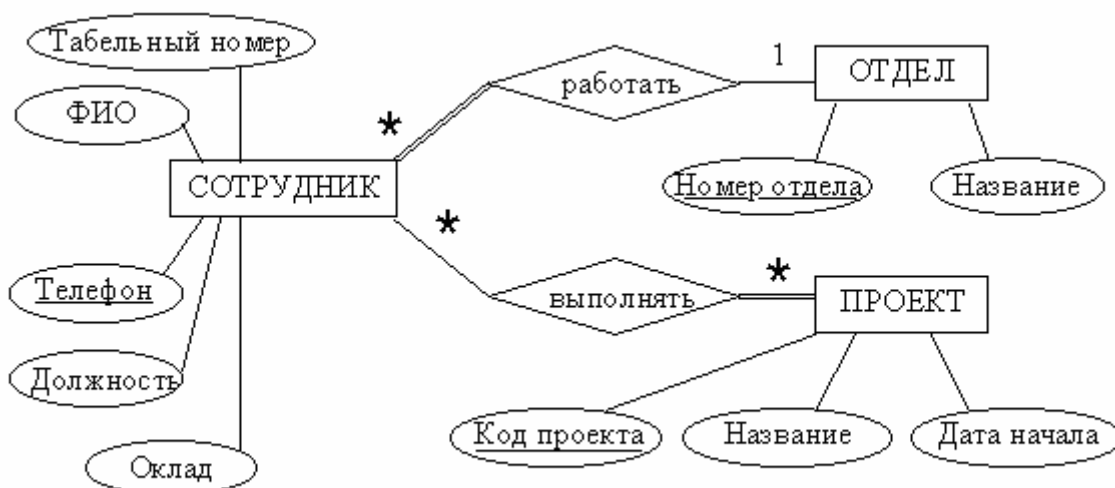
- полное (обязательное) участие объекта в связи – для существования некоторого объекта требуется существование другого объекта, связанного с первым связью (на диаграмме соединение с отношения с таким объектом выполняется двойной линией);
- частичное (необязательное) участие объекта в связи – для существования некоторого объекта не требуется существования другого объекта, связанного с первым связью.



Объект «Стройка» имеет неполную степень участия в отношении, значит он может не контролироваться инспектором; объект «Инспектор» имеет полную степень участия – обязательно должен иметь объект для контроля

Степень участия объектов в отношении.

Пример ER-диаграммы:



3. Порядок выполнения работы

1. По предложенному преподавателем заданию представить «реальный мир» (предметную область). То, что входит в эту предметную область – подлежит моделированию, то, что не входит – не подлежит. Для этого этапа допустимо словесное (умозрительное) представление данных. Задание формулируется только общим направлением (например, «библиотека», «столовая» и т.п.), т.к. моделирование предметной области также входит в задачи данной работы. Допустимо моделирование только некоторых аспектов

данных в предложенной области (например, только успеваемость школьников в направлении «школа» без учета других особенностей (например, турпоходов, олимпиад, школьной библиотеки и т.п.)).

2. Выделить единичные объекты предметной области (естественно не все, но так, чтобы «ассортимент» различных объектов был как можно больше).

3. Выделить классы объектов (подмножества качественно сходных объектов, число таких независимых классов объектов предметной области должно быть не менее 5 - 6).

4. Оценить, как могут быть взаимосвязаны между собой объекты разных классов.

5. Перевести классы объектов в сущности, свойства объектов, принадлежащих классам – в атрибуты. Выделить ключевые атрибуты сущностей.

6. Перевести связи между конкретными объектами в отношения между сущностями. Задать структурные ограничения для участников отношений. Полученную модель можно назвать концептуальной ER-диаграммой.

7. Оформить отчет, включающий в себя исходное задание и конечную концептуальную ER-диаграмму (допустимо указывать дополнительные пояснения, если семантика фрагментов диаграммы не ясна из названий).

ЛАБОРАТОРНАЯ РАБОТА №2

СОЗДАНИЕ РЕЛЯЦИОННОЙ МОДЕЛИ ДАННЫХ ПО ER-МОДЕЛИ

1. Цель работы

Познакомиться с реляционной моделью данных. Для указанного варианта задания представить ER-диаграмму в виде реляционной модели данных. Реализовать полученную реляционную модель данных для СУБД MS Access.

2. Краткие теоретические сведения

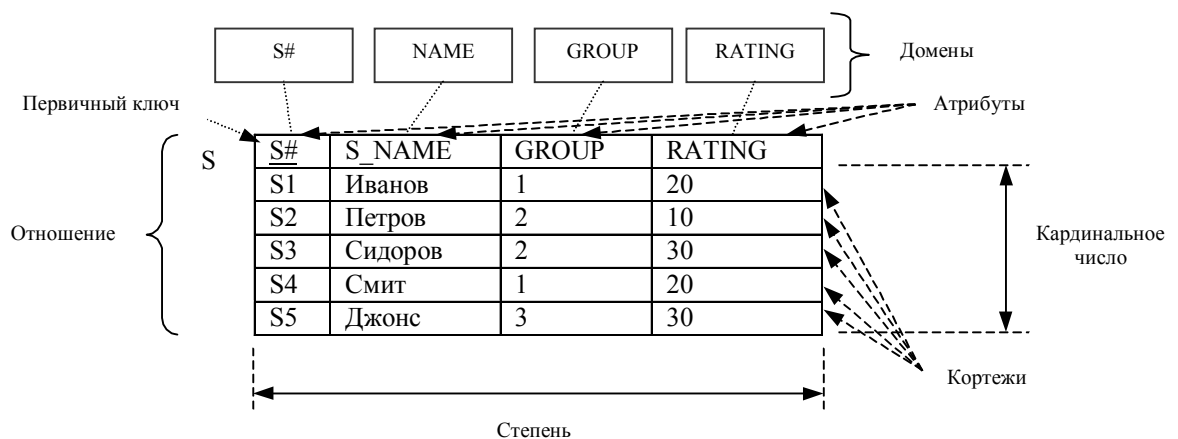
2.1. Реляционная модель данных

Реляционная модель – формальная теория данных, основанная на некоторых положениях математики (теории множеств и предикативной логике). Реляционная модель была предложена Э.Ф. Коддом (E.F. Codd, IBM, математик) в 1970 в статье «Реляционная модель данных для больших совместно используемых банков данных». Кодд впервые осознал, что математические дисциплины можно использовать, чтобы внести в область управления базами данных строгие принципы и точность (в то время не было устоявшихся терминов (однозначных определений) в этой области).

2.1.1. Реляционные объекты (структура модели)

Реляционная модель данных основана на математическом понятии отношения (relation), физическим представлением которого является таблица. Все данные (описания объектов) в реляционной БД пользователь воспринимает как набор таблиц (множество отношений). Реляционные объекты данных

(структура):



Реляционные объекты данных (отношение S).

Краткое (неформальное, т.е. нестрогое) описание терминов:

- отношение – плоская таблица;
- кортеж – строка таблицы (не включая заголовок);
- кардинальное число – количество строк таблицы (без заголовка);
- атрибут – столбец таблицы (или поле строки);
- степень – количество столбцов таблицы;
- первичный ключ – уникальный идентификатор для таблицы;
- домен – общая совокупность допустимых значений.

Описание основных структурных единиц модели:

1) Домены

Домен – именованное множество скалярных значений (скаляр – наименьшая (атомарная) семантическая единица данных (например, «номер детали»)) одного типа (например, в домене «города» будут представлены все возможные названия городов, которыми может оперировать модель).

Значения атрибута (атрибутов) выбираются из множества значений домена, при этом каждый атрибут должен быть определен на единственном

домене (т.е. в любой момент времени набор значений атрибута (в отношении) является некоторым подмножеством множества значений его домена, также в любой момент времени в домене могут существовать такие значения, которые не являются значениями ни одного из атрибутов, соответствующих этому домену).

Значение доменов в реляционной модели:

- домены позволяют централизованно определять смысл и источник значений, которые могут получать атрибуты (в системе не будет значений связанных с ошибками написания или непонимания семантики значения (вместо «да» или «нет» пользователю нельзя будет поставить прочерк));
- домены позволяют ограничивать сравнения (важнейшее свойство, позволяет избегать семантически некорректных операций), т.е. если значения атрибутов взяты из одного и того же домена, тогда операции над этими атрибутами (например, сравнение) будут иметь смысл, т.к. операция будет проводиться над между подобными по семантике атрибутами; если значения атрибутов взяты из различных доменов, то операции над ними, скорее всего, не будут иметь смысла (в основном предназначается для предотвращения грубых ошибок (механических опечаток или недостаточного знания модели)).

Домены имеют концептуальную природу. Для наличия возможности ограничения сравнений в системе нужно, чтобы домены, по крайней мере, были описаны в рамках определений БД, имели уникальные имена, и все атрибуты отношений имели бы ссылку на домен (при этом для упрощения системы наборы значений доменов могут в ней не храниться). На практике – большинство программных продуктов поддержки сравнений не обеспечивают, а смысл атрибутов и источник их значений определяется (иногда и только в случае небольшого количества значений) в других таблицах – не доменах, а их замене (например, названия всех дней недели определяются в одной таблице (содержит только один атрибут), а в другой таблице пользователю разрешен выбор значений атрибута только из значений первой таблицы).

2) Отношения

Отношение R , определенное на множестве доменов D_1, D_2, \dots, D_n (не обязательно различных), содержит две части: заголовок (строка заголовков столбцов в таблице) и тело (строки таблицы).

Заголовок содержит фиксированное множество атрибутов или пар вида $\langle \text{имя_атрибута} : \text{имя_домена} \rangle$:

$$\{ \langle A_1:D_1 \rangle, \langle A_2:D_2 \rangle, \dots, \langle A_n:D_n \rangle \},$$

причем каждый атрибут A_j соответствует одному и только одному из лежащих в основе доменов D_j ($j=1,2,\dots,n$). Все имена атрибутов A_1, A_2, \dots, A_n - разные.

Тело содержит множество кортежей. Каждый кортеж, в свою очередь, содержит множество пар $\langle \text{имя_атрибута} : \text{значение_атрибута} \rangle$:

$$\{ \langle A_1:v_{i1} \rangle, \langle A_2:v_{i2} \rangle, \dots, \langle A_n:v_{in} \rangle \},$$

($i=1,2,\dots,m$, где m – количество кортежей в этом множестве). В каждом таком кортеже есть одна пара такая пара $\langle \text{имя_атрибута} : \text{значение_атрибута} \rangle$, т.е. $\langle A_j:v_{ij} \rangle$, для каждого атрибута A_j в заголовке. Для любой такой пары $\langle A_j:v_{ij} \rangle$ v_{ij} является значением из уникального домена D_j , который связан с атрибутом A_j .

Значения m и n называются соответственно кардинальным числом и степенью отношения R .

Свойства отношений:

- отношение имеет уникальное имя (т.е. отличное от имен других отношений в БД);
- в любом отношении нет одинаковых кортежей (это свойство следует из того факта, что тело отношения – математическое множество (кортежей), а множества в математике по определению не содержат одинаковых элементов); важное следствие из этого свойства – т.к. отношение не содержит одинаковых кортежей, то всегда существует первичный ключ отношения;

- в любом отношении кортежи не упорядочены (т.е. нет упорядоченности «сверху-вниз») (это свойство следует из того, что тело отношения – математическое множество, а простые множества в математике неупорядочены); следствие – нет понятий позиции кортежа и последовательности кортежей в отношении, кортеж в отношении всегда определяется по первичному ключу;
- в любом отношении атрибуты не упорядочены (т.е. нет упорядоченности «слева-направо») (это свойство следует из того, что заголовок отношения также определен как множество атрибутов); следствие – атрибут всегда определяется по имени, нет понятий позиции атрибута и последовательности атрибутов в отношении;
- в отношении все значения атрибутов атомарные (неделимые) (это следует из того, что все лежащие в основе домены содержат только атомарные значения); следствие – в каждой ячейке на пересечении столбца и строки в таблице расположено только одно значение, отношения при этом представлены в первой нормальной форме (не содержат групп повторения) – простота структуры отношения приводит к упрощению всей системы (и облегчает операции с отношениями).

Как видно из свойств отношения, отношение и таблица на самом деле не одно и то же: таблица – конкретное представление (упорядоченное, обычно графическое) абстрактного объекта отношения.

Переменная отношения – именованное отношение, которое может иметь разные значения в разное время (однако все возможные значения этой переменной будут иметь одинаковые заголовки; этот термин предназначен для подчеркивания возможности изменения тела отношения, т.е. его динамичности (при этом термин «отношение» считается неудачным), при статичности описания отношения).

Различают множество видов отношений:

- именованное отношение – переменная отношения, определенная с помощью DDL в СУБД;

- базовое отношение – именованное отношение, которое является автономным (на практике – логически важные отношения, хранимые в БД (обычно соответствует некоторому объекту в концептуальной схеме));
- производное отношение – отношение, определенное посредством реляционного выражения через другие именованные отношения;
- выражаемое отношение – отношение, которое можно получить из набора именованных отношений посредством некоторого реляционного выражения (множество выражаемых отношений охватывает множество всех базовых и всех производных отношений);
- представление – именованное производное отношение (виртуально – представлено в системе только через определение в терминах других именованных отношений);
- снимок – именованное производное отношение, в отличие от представлений, снимок реален (не виртуален), т.е. имеет свои данные (отдельные от других именованных отношений);
- результат запроса – неименованное производное отношение, содержащее результат запроса (временное существование);
- промежуточный результат – неименованное производное отношение, являющееся результатом некоторого реляционного выражения, вложенного в другое, большее выражение;
- хранимое отношение – отношение, которое поддерживается в физической памяти.

В данной работе основными отношениями будут базовые отношения, представленные в БД таблицами.

3) Представления

Представление – динамический результат одной или нескольких реляционных операций над базовыми отношениями, является виртуальным отношением (т.е. реально в базе не существует, а содержит части реально существующих (базовых) отношений). Содержимое представления изменяется,

если изменяется содержимое частей базовых таблиц, связанных с данным представлением. Если пользователи вносят допустимые изменения в содержимое представления, то эти изменения «одновременно» вносятся и в базовые отношения. Представление описывает не всю внешнюю модель пользователя (представление пользователя), а только его некоторую часть. В данной работе представления использоваться не будут, т.к. каждый студент получает монопольный доступ к своей БД.

2.1.2. Целостность реляционных данных

Целостность данных предназначена для сохранения в БД «отражения действительности реального мира», т.е. устранения недопустимых конфигураций (состояний) значений и связей (например, вес детали может быть только положительным), которые не имеют смысла в реальном мире (не следует путать с «безопасностью» - хоть и есть некоторое сходство, но безопасность относится к защите от несанкционированного доступа, а целостность – защита БД от санкционированных пользователей).

Правила целостности данных можно разделить на:

- специфические или корпоративные ограничения целостности – правила, определяемые пользователями или администратором БД, которые указывают дополнительные ограничения, специфические для конкретных БД (например, максимальные и минимальные значения некоторых атрибутов (диапазон оценок));
- общие правила целостности – правила, которые применимы к любой реляционной БД (относятся к потенциальным (первичным) и к внешним ключам).

Описание основных концепций поддержки целостности в реляционной модели:

1) Потенциальные ключи

Пусть R – некоторое отношение. Тогда потенциальный ключ K для R – это подмножество множества атрибутов R , обладающее следующими свойствами:

- свойством уникальности – нет двух различных кортежей в отношении R (в текущем значении) с одинаковым значением K (каждое отношение имеет как минимум один потенциальный ключ, т.к. не содержит одинаковых кортежей, важно чтобы уникальность соблюдалась не только для конкретного содержания отношения (на данный момент времени), а всегда (т.е. для множества всех возможных значений отношения в любой момент времени) – для этого при определении потенциального ключа важно знать смысл атрибутов отношения);
- свойством неизбыточности (неприводимости) – никакое из подмножеств K не обладает свойством уникальности (если определенный «потенциальный ключ» на самом деле не является неизбыточным, то система не сможет обеспечить должным образом соответствующее ограничение целостности (например, если вместо номера студента $\{S\# \}$, потенциальный ключ будет определен как номер студента и номер группы $\{S\#, GROUP\}$, то уникальность номера студента будет поддерживаться системой в более узком (локальном) смысле – только в пределах одной группы)).

Простой потенциальный ключ – потенциальный ключ, состоящий из одного атрибута.

Составной потенциальный ключ – потенциальный ключ, состоящий более чем из одного атрибута.

Потенциальные ключи обеспечивают основной механизм адресации на уровне кортежей в реляционной системе (единственный гарантируемый системой способ точно указать некоторый кортеж – это указать значение некоторого потенциального ключа).

Отношение может иметь несколько (хотя это довольно редко

применяется) потенциальных ключей. По традиции (исторически) один из потенциальных ключей должен быть выбран в качестве первичного ключа отношения (отношение всегда имеет первичный ключ), а остальные потенциальные ключи будут называться альтернативными ключами. Выбор одного из потенциальных ключей в качестве первичного – вопрос философский и фактически выходит за рамки реляционной модели данных.

2) Внешние ключи

Пусть R_2 – базовое отношение. Тогда внешний ключ FK в отношении R_2 – это подмножество множества атрибутов R_2 , такое что:

- существует базовое отношение R_1 (при этом R_1 и R_2 не обязательно различны) с потенциальным ключом K ;
- каждое значение FK в текущем значении R_2 всегда совпадает со значением K некоторого кортежа в текущем значении R_1 (обратное не требуется, т.е. потенциальный ключ, соответствующий данному внешнему ключу, может содержать значение, которое в данный момент не является значением внешнего ключа).

Внешний ключ будет составным (т.е. будет состоять из более чем одного атрибута), тогда и только тогда, когда соответствующий потенциальный ключ также будет составным. Соответственно, внешний ключ будет простым тогда и только тогда, когда соответствующий потенциальный ключ также будет простым.

Каждый атрибут, входящий во внешний ключ, должен быть определен на том же домене, что и соответствующий атрибут соответствующего потенциального ключа.

Внешним ключом может быть любой атрибут (или сочетание атрибутов) отношения, а не только компонент потенциального ключа.

Значение внешнего ключа называется ссылкой к кортежу, содержащему соответствующее значение потенциального ключа (ссылочный кортеж или целевой кортеж). Отношение, которое содержит внешний ключ, называется

ссылающимся отношением, а отношением, которое содержит соответствующий потенциальный ключ – ссылочным или целевым отношением.

Для упрощения понимания, ссылки можно представлять с помощью ссылочных диаграмм – каждая стрелка обозначает внешний ключ в отношении, из которого эта стрелка выходит, этот ключ ссылается на первичный (потенциальный) ключ отношения, на который стрелка указывает (допустимо дополнительно указывать над стрелками атрибуты, которые составляют внешний ключ):

$$S \leftarrow SP \rightarrow P.$$

Отношение может быть одновременно ссылочным и ссылающимся (цепочка стрелок из R_n в R_1 называется ссылочный путь):

$$R_n \rightarrow R_{n-1} \rightarrow R_{n-2} \rightarrow \dots \rightarrow R_2 \rightarrow R_1.$$

Также выделяют самоссылающиеся отношения (в них возникают ссылочные циклы):

$$R_n \rightarrow R_{n-1} \rightarrow R_{n-2} \rightarrow \dots \rightarrow R_2 \rightarrow R_1 \rightarrow R_n.$$

3) Ссылочная целостность

Правило ссылочной целостности – база данных не должна содержать несогласованных значений внешних ключей (здесь «несогласованное значение внешнего ключа» - это значение внешнего ключа, для которого не существует отвечающего ему значения соответствующего потенциального ключа в соответствующем целевом отношении) – т.е. если В ссылается на А, тогда А должно существовать.

Понятия «внешний ключ» и «ссылочная целостность» определены в терминах друг друга, т.е. «поддержка внешних ключей» и «поддержка ссылочной целостности» означают одно и то же.

Для поддержки ссылочной целостности необходимо внести компенсацию в БД в случаях:

- при удалении объекта ссылки внешнего ключа (возможные действия: ограничить – запрет операции удаления объекта, до тех пор, пока на этот

объект есть ссылки; каскадировать – при удалении объекта удалить все объекты, ссылающиеся на него);

- при попытке обновить потенциальный ключ, на который ссылается внешний ключ (возможные действия: ограничить – запрет операции обновления объекта, до тех пор, пока на этот объект есть ссылки; каскадировать – при обновлении объекта обновить внешний ключ во всех объектах, ссылающиеся на него).

На практике, кроме перечисленных действий («ограничить» и «каскадировать»), должна быть возможность вызова процедуры баз данных (хранимой процедуры или процедуры триггеров), для частного решения возникшей проблемы.

С логической точки зрения, все операции с объектами БД (обновление и удаление) всегда атомарны (все или ничего), даже если они выполняются каскадно.

При рассмотрении реакции системы на операции с объектом (обновление или удаление) следует рассматривать все возможные ссылочные пути. Для самоссылающихся отношений некоторые проверки ограничений не могут быть выполнены во время одного обновления (они обычно откладываются на более позднее время – время фиксации).

4) NULL-значения

NULL-значения (определитель NULL) введены для обозначения значений атрибутов, которые на настоящий момент неизвестны или неприемлемы для некоторого кортежа – т.е. логическая величина «неизвестно». Это не значение по умолчанию, а отсутствие какого-либо значения (например, данные об адресе нового сотрудника неизвестны (на данный момент)).

Для каждого атрибута должно быть установлено, может ли он принимать NULL-значения или нет.

Применение определителя NULL может вызвать проблемы на этапе реализации системы, т.к. реляционная модель основана на исчислении

предикатов первого порядка (они обладают двузначной (булевой) логикой (т.е. могут иметь только два значения: «истина» или «ложь»)), а применение NULL-значений приводит к работе с логикой более высокого порядка (трехзначной или даже четырехзначной).

Использование NULL-значений – вопрос теоретически спорный. Не во всех реляционных системах поддерживается работа с определителем NULL.

NULL-значения влияют на концепции потенциальных и внешних ключей реляционной модели данных:

- целостность объектов – в реляционной модели данных ни один атрибут потенциального ключа (в некоторых источниках – только первичные ключи) базового отношения не может содержать NULL-значений (т.к. реляционная БД не должна хранить информацию о чем-то, чего мы не можем определить и однозначно сослаться).
- ссылочная целостность – если в реляционной модели в отношении существует внешний ключ, то значение внешнего ключа должно либо соответствовать значению потенциального ключа некоторого кортежа в целевом отношении, либо задаваться определителем NULL (здесь NULL-значение обозначает не «значение неизвестно», а – «значение не существует»).

Определение внешнего ключа с учетом NULL-значений:

Пусть R_2 – базовое отношение. Тогда внешний ключ FK в отношении R_2 – это подмножество множества атрибутов R_2 , такое что:

- существует базовое отношение R_1 (R_1 и R_2 не обязательно различны) с потенциальным ключом K;
- каждое значение FK в текущем значении R_2 или является NULL-значением, или совпадает со значением K некоторого кортежа в текущем значении R_1 .

При этом вносится дополнительное действие для поддержки ссылочной целостности в случаях:

- при удалении объекта ссылки внешнего ключа (дополнительное

действие: аннулировать – для всех ссылок на объект устанавливаются NULL-значения, а затем сам объект удаляется);

- при попытке обновить потенциальный ключ, на который ссылается внешний ключ (дополнительное действие: аннулировать – для всех ссылок на объект устанавливаются NULL-значения, а затем сам объект обновляется).

2.1.3. Получение реляционной модели из ER-диаграммы

Алгоритм преобразования ER-диаграммы в реляционную модель (схему) следующий:

1. Каждая простая сущность (объект на ER-диаграмме) превращается в таблицу. Простая сущность - сущность, не являющаяся подтипом и не имеющая подтипов. Имя сущности становится именем таблицы.

2. Каждый атрибут становится возможным столбцом с тем же именем; при этом может выбираться более точный формат. Столбцы, соответствующие необязательным атрибутам, могут содержать неопределенные значения; столбцы, соответствующие обязательным атрибутам, - не могут.

3. Компоненты уникального идентификатора сущности (уникальные атрибуты объекта) превращаются в первичный ключ таблицы. Если имеется несколько возможных уникальных идентификаторов, выбирается наиболее используемый. Если в состав уникального идентификатора входят связи, к числу столбцов первичного ключа добавляется копия уникального идентификатора сущности, находящейся на дальнем конце связи (этот процесс может продолжаться рекурсивно). Для именования этих столбцов используются имена концов связей и/или имена сущностей.

4. Связи многие-к-одному (и один-к-одному) становятся внешними ключами. Внешний ключ добавляется в виде столбца (столбцов) в таблицу, соответствующую объекту со стороны «многие» связи. Необязательные связи

соответствуют столбцам, допускающим неопределенные значения; обязательные связи - столбцам, не допускающим неопределенные значения.

5. Связи многие-ко-многим реализуются через промежуточную таблицу. Эта таблица будет содержать внешние ключи на соответствующие объекты, а также другие атрибуты, которые описывают указанную связь. Первичный ключ промежуточной таблицы должен включать в себя как минимум внешние ключи на объекты связи.

6. Если в концептуальной схеме присутствовали подтипы, то возможны два способа: (а) все подтипы в одной таблице и (б) для каждого подтипа - отдельная таблица. При применении способа (а) таблица создается для наиболее внешнего супертипа, а для подтипов могут создаваться представления. В таблицу добавляется по крайней мере один столбец, содержащий код ТИПА; он становится частью первичного ключа. При использовании метода (б) для каждого подтипа первого уровня (для более нижних - представления) супертип воссоздается с помощью представления UNION (из всех таблиц подтипов выбираются общие столбцы - столбцы супертипа).

7. Имеется два способа работы при наличии исключających связей: (а) общий домен и (б) явные внешние ключи. Если остающиеся внешние ключи все в одном домене, т.е. имеют общий формат (способ (а)), то создаются два столбца: идентификатор связи и идентификатор сущности. Столбец идентификатора связи используется для различения связей, покрываемых дугой исключения. Столбец идентификатора сущности используется для хранения значений уникального идентификатора сущности на дальнем конце соответствующей связи. Если результирующие внешние ключи не относятся к одному домену, то для каждой связи, покрываемой дугой исключения, создаются явные столбцы внешних ключей; все эти столбцы могут содержать неопределенные значения.

2.2. СУБД MS Access. Таблицы

MS Access поддерживает реляционную модель данных, т.е. база данных состоит из множества взаимосвязанных отношений, каждому отношению соответствует отдельная таблица. Таблицы состоят из строк и столбцов. Столбцы таблицы отражают общие свойства или характеристики объектов реального мира, которые моделируются с использованием данной таблицы. Для каждого свойства пользователь выделяет один столбец. Строки таблицы соответствуют различным экземплярам реальных объектов и называются записям.

2.2.1. Создание таблицы в MS Access

Прежде чем создать таблицу, необходимо открыть базу данных, в которой MS Access будет хранить эту таблицу. Затем в окне базы данных следует выбрать закладку «Таблицы» и в окне выбрать необходимый режим создания таблицы. В лабораторной работе рекомендуется использовать режим «Создание таблицы в режиме конструктора». В режиме конструктора можно изменить имеющиеся поля и добавить новые. Если таблица уже содержит данные, то при изменении структуры таблицы они теряются только в исключительных случаях. MS Access предупреждает об этом соответствующим сообщением.

Конструктор позволяет описать заголовок таблицы, задав свойства каждого столбца. Для каждого столбца обязательно необходимо задать свойства «Имя поля» и «Тип данных».

Рекомендуется задавать имена полей не искажающие смысловое значение данных поля. В именах полей запрещено использовать символы '!', '.', '[', ']'. Максимальная длина имени 64 символа. Не рекомендуется использовать в именах полей пробелы и символы русского (белорусского) алфавита, т.к. в

дальнейшем могут некорректно выполняться запросы на SQL.

Свойство “Тип данных” (DataType) для каждого столбца создаваемой таблицы определяет тип данных, хранящихся в поле таблицы. Допустимы следующие значения свойства “Тип данных” (табл. 1).

Таблица 1

Значение	Описание
Текстовый (Text)	Текст, максимальная длина которого равняется 255 символам или определяется значением свойства “Размер поля” (FieldSize) (используется по умолчанию)
Поле MEMO (Memo)	Текст, максимальная длина которого равняется 64 000 байтам. Поля этого типа не могут быть индексированными
Числовой (Number)	Любые числа. Для получения более подробных сведений смотрите описание свойства “Размер поля”.
Дата/время (Date/Time)	Даты и время, относящиеся к годам с 100 по 9999, включительно.
Денежный (Currency)	Числа, хранящиеся с точностью до 15 знаков в целой и до 4 знаков в дробной части.
Счетчик (AutoNumber)	Число, автоматически увеличиваемое при добавлении в таблицу каждой новой записи. Поля этого типа не подлежат изменению.
Логический (Yes/No)	Логические значения, а также поля, которые могут содержать одно из двух возможных значений. Поля этого типа не могут быть индексированными.
Поле объекта OLE (OLE Object)	Объект (например, рисунок Microsoft Draw) созданный другим приложением. Максимальный размер – примерно один гигабайт. Поля этого типа не могут быть индексированными.

Для каждого поля таблицы необходимо обязательно задать тип данных. В каждом поле могут храниться данные только одного типа. Изменение типа поля после ввода данных в таблицу может привести к потере данных.

Для поля можно ввести «Описание» или произвольный комментарий, относящийся к полю данных. Типичным комментарием является детальное описание назначения поля.

Дополнительные свойства полей:

1. “Размер поля” (FieldSize) - задает максимальный размер данных, которые могут быть помещены в данное поле. Рекомендуется использовать как можно меньшее значение свойства “Размер поля”, поскольку обработка данных меньшего размера выполняется быстрее и требует меньше памяти. Преобразование большего значения свойства “Размер поля” к меньшему в таблице, которая уже содержит данные, может привести к потере данных.

2. “Обязательное поле” (Required) - указывает, можно ли вводить в данное поле пустые (Null) значения.

3. “Пустые строки” (AllowZeroLength) - указывает, можно ли вводить в данное поле строки нулевой длины (“”). Допустимы следующие значения свойства “Пустые строки” (табл. 2). Для того чтобы ввести в поле строку нулевой длины, введите две кавычки(“”). Значения свойств “Пустые строки” и “Обязательное поле” можно использовать для различения несуществующих данных (хранящихся в виде строк нулевой длины) и данных, которые существуют, но неизвестны (хранящихся в виде пустых (Null) значений). Если свойство “Пустые строки” имеет значение “Да”, значит, в данное поле можно вводить строки нулевой длины, независимо от значения свойства “Обязательное поле”.

Таблица 2

Значение	Описание
Да	В данное поле можно вводить строки нулевой длины.
Нет	В данное поле нельзя вводить строки нулевой длины (используется по умолчанию).

4. “Заголовок” (Caption) - указывает текст, который отображается в связанной с полем подписи и используется в качестве заголовка столбца в режиме таблицы. Длина подписи поля, формы или кнопки может достигать 255 символов. Подпись поля можно определить в окне конструктора таблиц или в окне запроса (в списке “Свойства поля”). Обычно подписи используются для отображения полезных сведений. Если подпись связанного поля не определена, то в качестве нее используется имя базового поля.

5. “Индексированное поле” (Indexed) - определяет индекс по одному полю. Допустимы следующие значения свойства “Индексированное поле” (табл. 3). Это свойство можно определить в бланке свойств в окне конструктора таблиц (в списке “Свойства поля”). Кроме того, можно выбрать команду Индексы в меню ВИД или нажать кнопку “Индексы” на панели инструментов. На экране появится окно “Индексы”. После добавления индекса по одному полю в окно “Индексы” свойство “Индексированное поле” автоматически примет значение “Да”.

Таблица 3

Значение	Описание
Нет	Не создает индекс по данному полю (используется по умолчанию).
Да (Допускаются совпадения)	Создает индекс по данному полю.
Да (Совпадения не допускаются)	Создает уникальный индекс по данному полю.

Используйте свойство “Индексированное поле” для ускорения выполнения поиска и сортировки записей по одному полю таблицы. Индексированное поле может содержать как уникальные, так и повторяющиеся значения. Пользователь может создать сколько угодно индексов. Индексы создаются при сохранении макета таблицы и автоматически обновляются при вводе и изменении записей. Пользователь может в любое время добавить новые

или удалить ненужные индексы в окне конструктора таблиц. Если ключ таблицы состоит из одного поля, то MS Access автоматически устанавливает значение “Да (Совпадения не допускаются)” свойства “Индексированное поле” для данного поля. Однако индексы замедляют изменение, ввод и удаление данных, поэтому не рекомендуется создавать избыточные индексы. Примечание: МЕМО, логические и OLE-поля не могут быть индексированными. Для создания составных индексов следует использовать окно “Индексы”.

2.2.2. Связывание таблиц

Для реляционных СУБД данные разных категорий хранятся в разных таблицах. Это позволяет исключить избыточность информации.

На практике наиболее часто встречается связь “один-ко-многим” (1:M). При таком типе связи каждой записи главной таблицы могут быть поставлены в соответствие одна или несколько записей, так называемой, подчиненной таблицы.

Для создания связи необходимо определить первичный ключ, как для главной, так и для подчиненной таблиц. Определение первичного ключа для подчиненной таблицы хотя и не является обязательным, но значительно увеличивает скорость работы. При создании таблиц с помощью Конструктора MS Access автоматически задает первичный ключ (в этом случае поле первичного ключа таблицы имеет тип данных "Счетчик" (AutoNumber) -этот тип данных гарантирует, что во время ввода данных MS Access автоматически нумерует строки таблицы в возрастающей последовательности (в ходе выполнения лабораторной работы не желательно использовать счетчик как первичный ключ таблицы, т.к. он обычно не несет семантический смысл)). Чтобы определить в качестве первичного ключа иной набор полей, следует выделить соответствующие поля и нажать пиктограмму «Ключевое поле» на

панели инструментов. Во время ввода данных в поля, определенные в качестве первичного ключа, MS Access автоматически следит за тем, чтобы вводились только уникальные значения. MEMO-поля и поля объекта OLE не могут быть первичными ключами.

Для подчиненной таблицы надо определить поле внешнего ключа, тип данных и размер, которые совпадают с полем первичного ключа главной таблицы. Внешние ключи отличаются от первичных тем, что для них допускаются наличие одинаковых значений полей.

MS Access автоматически следит за обеспечением целостности данных. Если редактировать запись в подчиненной таблице, то эта запись может быть сохранена лишь в том случае, если значение связующего поля присутствует в главной таблице. При редактировании главной таблицы можно удалить запись лишь в том случае, если эта запись не связана с записями подчиненной таблицы.

Таблицы связываются в режиме формирования схемы БД, при этом для установления связи 1:М необходимо установить мышью на первичный ключ главной таблицы и потом протащить линию связи до подчиненной таблицы. Характеристики связи задаются дополнительно.

2.2.3. Дополнительные режимы работы с базой данных

При практической эксплуатации базы данных, созданной с использованием MS Access возникает необходимость общего копирования БД, восстановления БД после сбоев, сжатия БД и наконец репликации – режима получения текущей копии с возможностью эксплуатации ее на переносном или изолированном компьютере с последующей синхронизацией обеих копий БД.

Сжатие БД необходимо, когда Вы очень интенсивно модифицируете БД либо в процессе разработки новых объектов (таблиц, форм, отчетов, запросов, модулей) либо в процессе добавления и удаления данных. В этих случаях все

удаляемые объекты и данных удаляются логически, т.е. физически расположены внутри файла *.mdb, который разрастается значительно и более сложно администрируется. Именно для этого и используется режим сжатия. Операция сжатия проводится над закрытой БД, рекомендуется сначала присвоить сжатой БД новое имя и только после благополучного завершения операции уничтожить старую копию и переименовать сжатую БД. Для проведения операции сжатия необходимо выбрать меню «Сервис» > «Служебные программы» > «Сжать и восстановить базу данных...».

Кроме того, в службе «Сервис» находятся еще разделы, связанные с различными настройками. К ним относятся:

1. Разделение БД на 2 взаимосвязанные части, в одной из которых находятся все таблицы, а в другой объекты их обработки: формы, запросы, отчеты, модули. Такое разделение обеспечивает наиболее эффективный способ эксплуатации системы, так как модификация объектов обработки может производиться отдельно, и оба файла связываются только с использованием специального режима надстройки Диспетчера связанных таблиц.

2. Диспетчер кнопочных форм, который обеспечивает быструю и гибкую разработку внешнего интерфейса системы, определяющего перечень основных режимов работы и их иерархию.

Подсоединение таблиц к базе данных можно осуществить и иным способом: в меню «Файл» > «Внешние данные» > «Связь с таблицами». Кроме того, можно импортировать таблицы целиком из других баз данных, созданных в MS Access и даже из других баз данных в иных форматах, однако при импорте таблица преобразуется и копируется целиком в текущую БД.

Более подробную информацию по использованию MS Access можно используя меню «Справка».

3. Порядок выполнения работы.

1. Проанализировать исходную ER-диаграмму и используя выше приведенный алгоритм получить реляционную модель данных.
2. Реализовать полученные реляционные отношения в виде таблиц для СУБД MS Access (задать свойства полей).
3. Задать первичные ключи таблиц (может потребовать коррекции свойств ключевых полей).
4. Задать внешние ключи таблиц и составить схему данных.
5. Определить типы связей между таблицами и задать целостность данных (каскадное удаление и обновление).
6. Проверить полученную схему путем внесения в таблицы данных (1-3 строки на таблицу).
7. Оформить отчет, включающий исходную ER-диаграмму и конечную схему базы данных.

ЛАБОРАТОРНАЯ РАБОТА №3

НОРМАЛИЗАЦИЯ РЕЛЯЦИОННОЙ МОДЕЛИ ДАННЫХ

МЕТОДОМ ДЕКОМПОЗИЦИИ

1. Цель работы

Познакомиться с функциональными зависимостями и нормальными формами реляционных схем. Для указанного варианта задания выполнить нормализацию отношений методом декомпозиции с учетом выделенных функциональных зависимостей. Представить результат в виде нормализованной реляционной модели.

2. Краткие теоретические сведения

Нормализация – метод создания набора отношений с заданными свойствами на основе некоторых требований к данным. Процесс нормализации предложен Коддом (1972 г.). Процесс нормализации – формальный метод для оптимизации столбцов отношений и устранения аномалий.

2.1. Избыточность данных и аномалии обновления

Основная цель проектирования реляционной БД – группирование атрибутов в отношениях таким образом, чтобы минимизировать избыточность данных (сокращение объема вторичной памяти для хранения БД) и повышение надежности при работе с данными. Обычно процесс проектирования отношений реляционной БД ведется на основе разработанной ER-диаграммы или на основе просто здравого смысла разработчика. В общем случае при таком

подходе расположение атрибутов в отношениях неоптимальное.

При работе с отношениями, содержащими избыточные данные, могут возникать проблемы – аномалии обновления. Аномалии обновления делят на три вида:

1) Аномалии вставки – возникают при добавлении новых несогласованных данных (нарушающих целостность данных в отношении).

2) Аномалии изменения – возникают при изменении части ранее введенных данных; частичное обновление сведений приведет к нарушению целостности данных отношения (перестанет выполняться оговоренная выше зависимость).

3) Аномалии удаления – возникают при удалении строк из отношений.

Интуитивно понятно, что решением проблем избыточности и аномалий для выше приведенного примера может стать разделение отношения на такие отношения, в которых избыточности не будет. Для выполнения такого процесса необходимо выявить все зависимости между атрибутами отношения (потеря одной такой зависимости меняет модель внешнего мира).

2.2. Функциональные зависимости

Выявление смысловой зависимости между данными – один из способов формализации смысловой информации о данных.

Функциональная зависимость описывает связь типа многие-к-одному между атрибутами отношения, где много – детерминант функциональной зависимости. Функциональная зависимость является семантическим свойством атрибутов отношения.

Если в отношении R , содержащем атрибуты A и B , атрибут B

функционально зависит от атрибута А (А является детерминантом атрибута В)

$$A \rightarrow B,$$

то в каждом кортеже этого отношения каждое конкретное значение атрибута А всегда связано только с одним значением атрибута В. Атрибуты А и В могут быть составными атрибутами.

Особенности функциональных зависимостей, лежащие в основе процесса нормализации:

- функциональная зависимость является специализированным правилом целостности – она накладывает ограничения на допустимые значения атрибутов отношений; эту особенность можно использовать при обновлении БД, т.к. зная какие функциональные зависимости есть в отношении, можно проанализировать нарушат ли новые данные целостность данных отношения;
- функциональная зависимость является обобщением понятия потенциального ключа; функциональные зависимости позволяют определить все потенциальные ключи отношения (и соответственно – первичный ключ): все атрибуты отношения, которые не являются частью первичного (или потенциального) ключа, должны функционально зависеть от этого ключа; если не все остальные атрибуты отношения зависят от некоторого детерминанта, то этот детерминант не является потенциальным ключом этого отношения.

Одни функциональные зависимости подразумевают другие зависимости. Для данного множества зависимостей S замыканием S^+ называется множество всех функциональных зависимостей, подразумеваемых зависимостями множества S . Множество S обязательно является подмножеством собственного замыкания S^+ . Правила логического вывода Армстронга (аксиомы Армстронга) обеспечивают исчерпывающую и полную основу для вычисления замыкания S^+ для заданного множества S :

- рефлексивность $X \rightarrow X$;

- пополнение $X \rightarrow Y \Rightarrow XZ \rightarrow Y$;
- аддитивность $(X \rightarrow Y) \text{ and } (X \rightarrow Z) \Rightarrow X \rightarrow YZ$;
- проективность $X \rightarrow YZ \Rightarrow X \rightarrow Y$;
- транзитивность $(X \rightarrow Y) \text{ and } (Y \rightarrow Z) \Rightarrow X \rightarrow Z$;
- псевдотранзитивность $(X \rightarrow Y) \text{ and } (YZ \rightarrow W) \Rightarrow XZ \rightarrow W$.

Два множества функциональных зависимостей S1 и S2 эквивалентны тогда и только тогда, когда они являются покрытиями друг друга, то есть $S1^+ = S2^+$.

Каждое множество функциональных зависимостей эквивалентно, по крайней мере, одному неприводимому множеству. Множество функциональных зависимостей является неприводимым, если:

- каждая функциональная зависимость этого множества имеет одноэлементную правую часть;
- ни одна функциональная зависимость множества не может быть устранена без изменения замыкания этого множества;
- ни один атрибут не может быть устранен из левой части любой функциональной зависимости без изменения замыкания множества.

Если I является неприводимым множеством, эквивалентным множеству S, то проверка выполнения функциональных зависимостей из множества I автоматически проверяет выполнение всех функциональных зависимостей множества S.

2.3. Нормальные формы и схемы выполнения нормализации

Нормализация – это формальный метод анализа отношений на основе их первичного ключа и существующих функциональных зависимостей.

В процессе нормализации реляционных отношений применяются концепции нормальных форм. Говорят, что отношение находится в определенной нормальной форме, если оно удовлетворяет правилам этой нормальной формы. В настоящее время используется шесть нормальных форм, которые зависят друг от друга путем усложнения (вложенности) набора правил:

$$1НФ \rightarrow 2НФ \rightarrow 3НФ \rightarrow НФБК \rightarrow 4НФ \rightarrow 5НФ.$$

Каждая нормальная форма, таким образом, удовлетворяет всем предыдущим нормальным формам. Более высокая нормальная форма приводит к более строгому формату отношения (меньшее число аномалий обновления).

БД можно построить и на отношениях находящихся в первой нормальной форме, но такая БД будет сильно подвержена аномалиям и избыточности данных. На практике желательно использовать как минимум 3НФ, чтобы устранить большинство аномалий обновления. Следует отметить, что процесс нормализации обратим (денормализация), то есть всегда можно использовать его результат для обратного преобразования, т.к. в процессе нормализации не утрачиваются первоначальные функциональные зависимости.

Определения нормальных форм:

1) 1НФ. Отношение находится в 1НФ тогда и только тогда, когда в любом допустимом значении этого отношения каждый кортеж содержит только одно значение для каждого из атрибутов, т.е. это значение не имеет внутренней структуры (множество, таблица и т.п.). отношения в 1НФ имеют большое количество аномалий обновления, для поддержания целостности БД требуется разработка сложных триггеров.

2) 2НФ. Отношение находится в 2НФ тогда и только тогда, когда оно находится в 1НФ, и каждый атрибут отношения не входящий в состав первичного ключа характеризуется полной функциональной зависимостью от

этого первичного ключа. Полной функциональной зависимостью называется такая зависимость $A \rightarrow B$, когда B функционально зависит от A и не зависит ни от какого подмножества A (т.е. удаление какого-либо атрибута из A приведет к утрате этой функциональной зависимости). 2НФ устраняет в отношении частичные функциональные зависимости неключевых атрибутов от первичного ключа, которые выносятся в отдельное отношение вместе с копиями своих детерминантов.

3) 3НФ. Отношение находится в 3НФ тогда и только тогда, когда оно находится в 2НФ и не имеет не входящих в первичный ключ атрибутов, которые находились бы в транзитивной функциональной зависимости от этого первичного ключа. Транзитивной функциональной зависимостью называется зависимость $A \rightarrow C$, если существуют зависимости $A \rightarrow B$ и $B \rightarrow C$ (говорят что атрибут C транзитивно зависит от A через атрибут B), при условии, что атрибут A функционально не зависит ни от атрибута B , ни от атрибута C . 3НФ устраняет в отношении транзитивные функциональные зависимости неключевых атрибутов от первичного ключа, которые выносятся в отдельное отношение вместе с копиями своих детерминантов. В 3НФ устранено большинство аномалий от первичного ключа, но отношение в этой форме имеет аномалии в случае наличия более чем одного потенциального ключа.

4) НФБК (нормальная форма Бойса-Кодда). Отношение находится в НФБК тогда и только тогда, когда каждый его детерминант является потенциальным ключом. Нарушения требований НФБК случаются, если в отношении есть два и более составных потенциальных ключа и эти ключи перекрываются (совместно используют хотя бы один общий атрибут). Для отношения с единственным потенциальным ключом, НФБК и 3НФ эквивалентны. В НФБК устраняются аномалии связанные с функциональными зависимостями не от потенциальных ключей отношения.

5) 4НФ. Отношение находится в 4НФ тогда и только тогда, если все его многозначные зависимости, которым оно удовлетворяет, являются функциональными зависимостями от потенциальных ключей этого отношения. Многозначной зависимостью называется такая зависимость между атрибутами А и В некоторого отношения, при которой для каждого значения атрибута А существуют соответствующие наборы (множество) значений атрибута В. В 4НФ устраняются многозначные зависимости, которые не являются функциональными зависимостями.

6) 5НФ. Отношение находится в 5НФ тогда и только тогда, когда каждая нетривиальная зависимость соединения в отношении подразумевается его потенциальными ключами. Зависимостью соединения называется такая зависимость, при которой декомпозиция (разделение) отношения может сопровождаться генерацией ложных строк при выполнении обратного соединения декомпозированных отношений путем выполнения операции естественного соединения. В 5НФ устраняются зависимости соединения, которые не обусловлены потенциальными ключами.

Способы выполнения нормализации можно разделить на:

1) Синтез – получение БД в 3НФ путем компоновки всех заведомо известных функциональных зависимостей с использованием нахождения неприводимого минимального покрытия I множества функциональных зависимостей S. Метод синтеза не требует построения начальной логической схемы БД, однако сложен для выполнения, т.к. число функциональных зависимостей всей БД может быть большим (сложность формирования I). Достоинства метода синтеза – оптимальное число отношений БД, все отношения сразу находятся в 3НФ.

2) Декомпозицию – формирование отношений БД путем разделения их на более мелкие, если эти отношения не выполняют правила необходимой

нормальной формы. Процесс декомпозиции имеет два свойства:

- соединение без потерь – восстановление любого кортежа исходного отношения, используя соединение кортежей отношений, полученных в результате декомпозиции;
- сохранение зависимостей – функциональные зависимости при декомпозиции сохраняются.

Для выполнения процесса декомпозиции в начале необходимо построение исходной концептуальной модели БД (например, ER-диаграммы), которую преобразуют в начальные ненормализованные отношения. К недостаткам нормализации путем декомпозиции относят:

- временная сложность – неполиномиальна и определяется полным перебором всех порождаемых отношений (это число заранее не известно);
- число полученных отношений может быть больше оптимального для 3НФ;
- возможны потери либо порождение новых функциональных зависимостей (характерно для более высоких нормальных форм);

Достоинства метода декомпозиции:

- разделение задачи на подзадачи, что позволяет выполнять задачу параллельно и с меньшей нагрузкой (и соответственно с меньшим числом ошибок);
- получение отношений в любой нормальной форме в любых сочетаниях.

Суть процесса нормализации:

1) В нормализованных отношениях не разрешаются никакие функциональные зависимости, кроме функциональных зависимостей вида $K \rightarrow A$, где K – потенциальный ключ отношения R , а A – неключевой атрибут.

2) Если же отношение R имеет функциональные зависимости $B \rightarrow A$, где

В не является потенциальным ключом, то в отношении R будет наблюдаться избыточность данных.

Выполнение нормализации важно не только при первичном проектировании реляционной БД, но также и при корректировке модели данных в процессе эксплуатации БД для учета новых функциональных зависимостей и устранения внесенных аномалий.

3. Порядок выполнения работы

1. Выделить функциональные зависимости для каждого отношения исходной реляционной схемы. Проверить практический смысл выделенных функциональных зависимостей.

2. Для каждого отношения (включая и вновь создаваемые) последовательно применить правила нормальных форм. При несоблюдении текущего правила в отношении, выполнить его декомпозицию (удалить проблемный атрибут из отношения с образованием нового отношения, первичным ключом которого будет детерминант рассматриваемой функциональной зависимости (этот атрибут только копируется)). Нормализованное отношение должно удовлетворять как минимум 3НФ (для каждого отношения должен быть задан первичный ключ).

3. Для полученной нормализованной реляционной схемы проверить смысл ссылок.

4. Реализовать полученные реляционные отношения в виде таблиц для СУБД MS Access (подробнее см. лабораторную работу №2).

7. Оформить отчет, включающий исходную и нормализованную схему базы данных.

ЛАБОРАТОРНАЯ РАБОТА №4

РАБОТА С РЕЛЯЦИОННОЙ БД НА ЯЗЫКЕ SQL

1. Цель работы.

Познакомиться с операторами языка SQL, отвечающими за выборку, добавление, модификацию и удаление данных из БД. Для указанного варианта задания представить запросы на языке SQL, реализованные для СУБД MS Access.

2. Краткие теоретические сведения

2.1. Язык DML SQL92

2.1.1. Оператор выборки данных SELECT

Оператор SELECT предназначен для выборки и отображения нужным образом данных БД.

2.1.1.1. Общий формат оператора выборки

```
SELECT [DISTINCT | ALL] { * | [column [AS new_column_name]] [...]}  
FROM table [alias] [...]  
[WHERE condition]  
[GROUP BY list [HAVING condition]]  
[ORDER BY list]
```

где column – имя столбца (или константа, или выражение);

DISTINCT – результат не будет содержать строк-дубликатов;

ALL – результат может содержать дублирующие строки (по

умолчанию);

- * - все столбцы;
- table - имя таблицы;
- alias - сокращение для имени таблицы;
- condition - условие фильтрации строк данных;
- list - список столбцов;

Выражения могут состоять из имен столбцов, констант, скобок('(', ')'), скалярных операторов (например: +, -, *, /).

Пример 1: простой запрос:

```
SELECT column1 AS koeff, '%', column1*100 AS procent  
FROM t1;
```

t1 : таблица				
	column1	column2	column3	column4
▶	0.3	1	AAA	AAA
	0.4	2	BBB	
	0.9	3	AAB	
	0.25	4	BAB	
*	0	0		

а) исходные данные

z1 : запрос на выборку			
	koeff	Expr1001	procent
▶	0.3	%	30.000001192
	0.4	%	40.000000596
	0.9	%	89.999997616
	0.25	%	25
*	0		

б) результат выполнения запроса

2.1.1.2. Секция WHERE - фильтр строк данных по условию

Строки, для которых условие фильтра выполняется, попадают на

дальнейшую обработку!

Основные типы условий:

1) сравнение – использование операторов сравнения (>, <, >=, <=, =, <>), скобок ('(', ')'), логических связок (AND («и»), OR («или»), NOT («нет»)) и констант.

2) принадлежность к множеству – использование [NOT] IN (value_list);

3) соответствие шаблону – использование [NOT] LIKE 'template' (специальные символы шаблона по стандарту: '%' – любая последовательность символов; '_' – любой одиночный символ; для поиска символов '%' и '_' можно задать ESCAPE символ (например, для поиска по шаблону '15%' можно задать LIKE '15#%' ESCAPE '#'));

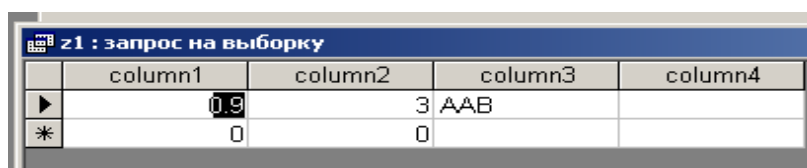
4) неизвестные значения (NULL) – использование IS NULL или NOT NULL.

Пример 2: использование фильтра WHERE (! обратите внимание на шаблон LIKE – особенности синтаксиса MS Access (? = _, * = %)):

```
SELECT *
```

```
FROM t1
```

```
WHERE (column1>0.2) And (column2 IN (1,2,3)) And (column3 LIKE '?A*')  
AND (column4 IS NULL);
```



	column1	column2	column3	column4
▶	0.9	3	AAB	
*	0	0		

Результат выполнения запроса (см. исходные данные к примеру 1)

2.1.1.3. Обобщающие (агрегатные) функции

Обобщающие функции предназначены для работы со столбцами (столбец представляется как множество данных). Обобщающие функции используются в

секциях SELECT и HAVING запроса. Все функции игнорируют NULL значения. Не допускается вложение обобщающих функций друг в друга.

Варианты обобщающих функций:

COUNT(column) – количество значений в столбце column (NULL значения игнорируются);

COUNT(*) - количество строк в таблице;

SUM(column) - сумма значений по столбцу column;

AVG(column) - среднее значение по столбцу column;

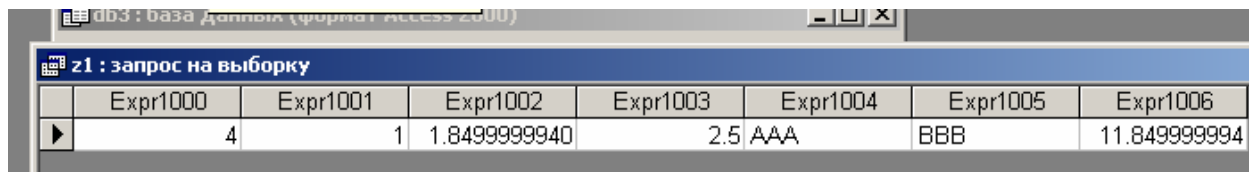
MIN(column) - минимальное значение по столбцу column;

MAX(column) - максимальное значение по столбцу column;

Запись DISTINCT в скобках перед именем столбца устраняет значения-дубликаты.

Пример 3: использование обобщающих функций в простых запросах:

```
SELECT COUNT(*), COUNT(column4), SUM(column1), AVG(column2),  
MIN(column3), MAX(column3), SUM(column1+column2)  
FROM t1;
```



z1 : запрос на выборку						
Expr1000	Expr1001	Expr1002	Expr1003	Expr1004	Expr1005	Expr1006
4	1	1.8499999940	2.5	AAA	BBB	11.849999994

Результат выполнения запроса (см. исходные данные к примеру 1)

2.1.1.4. Группирующие запросы (секция GROUP BY)

Группирующие запросы служат для обобщения (группировки) результатов по одинаковым значениям (группам) в указанных после GROUP BY столбцах. Для каждой группы в таком запросе создается только одна строка результата. Все имена столбцов, приведенные в описании SELECT должны обязательно присутствовать и в секции GROUP BY. Если совместно с GROUP

BY используется WHERE, то WHERE обрабатывается первым, а группированию подвергаются только те строки, которые удовлетворяют условию фильтра. По ISO, NULL-значения входят в одну группу.

Для фильтрации данных группы по заданным условиям используется подгруппа HAVING.

Пример 4: группирующий запрос:

```
SELECT column2, COUNT(*), COUNT(column3), AVG(column1)
FROM t1
WHERE column1 > 0.4
GROUP BY column2 HAVING COUNT(column3) > 1;
```

z1 : запрос на выборку

t1 : таблица				
	column1	column2	column3	column4
	0.5	1	AAA	AAA
	0.6	1	AAA	
	0.7	1	AAB	
	0.8	2	BAB	
	0.9	2	BAB	
	0.4	2	DAA	
	0.6	2	AQQ	
	0.8	3	DEW	
	0.5	4	AQQ	
	0.7	4	FSD	
	0.4	4	FFA	
*	0	0		

а) исходные данные

z1 : запрос на выборку

	column2	Expr1001	Expr1002	Expr1003
	1	3	3	0.600000004
▶	2	3	3	0.7666666706
	4	2	2	0.5999999940

б) результат выполнения запроса

2.1.1.5. Секция ORDER BY – сортировка результатов запроса

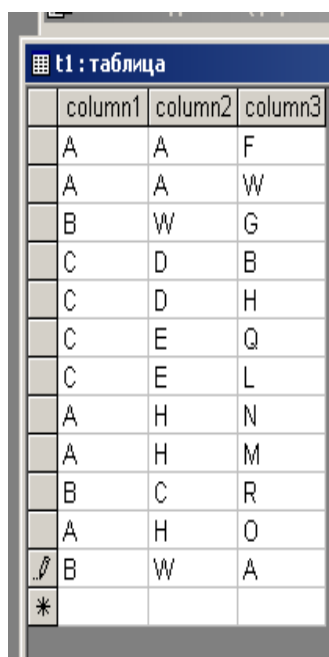
Секция ORDER BY определяет упорядоченность результатов по столбцам. Первый столбец в списке называется главным ключом и определяет общую упорядоченность строк результирующей таблицы. Последующие столбцы в списке сортировки определяют дополнительное упорядочивание в общей упорядоченности. Порядок сортировки данных столбца задается после имени столбца: ASC – сортировка по возрастанию значений (используется по умолчанию); DESC – сортировка по убыванию значений. По ISO, NULL-значения либо наибольшее, либо наименьшее (на усмотрение разработчиков СУБД).

Пример 5: использование сортировки:

SELECT *

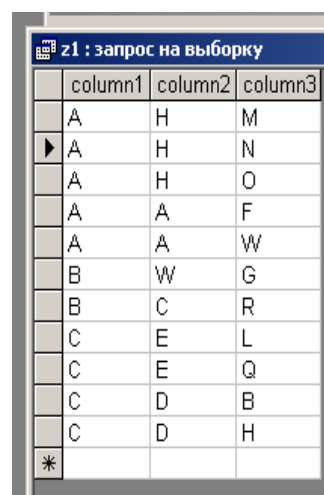
FROM t1

ORDER BY column1, column2 DESC, column3;



column1	column2	column3
A	A	F
A	A	W
B	W	G
C	D	B
C	D	H
C	E	Q
C	E	L
A	H	N
A	H	M
B	C	R
A	H	O
B	W	A
*		

а) исходные данные



column1	column2	column3
A	H	M
A	H	N
A	H	O
A	A	F
A	A	W
B	W	G
B	C	R
C	E	L
C	E	Q
C	D	B
C	D	H
*		

б) результат выполнения запроса

2.1.1.6. Подзапросы

Подзапросы – запросы с помощью оператора SELECT, помещенные в

секции WHERE и (или) HAVING внешнего оператора SELECT. Подзапрос создает временную таблицу, содержимое которой извлекается и обрабатывается внешним оператором (обычно предикатом внешнего запроса). Текст подзапроса должен быть заключен в круглые скобки и располагается в правой части операции внешнего запроса. В подзапросах не должна использоваться секция ORDER BY.

Подзапрос может вернуть следующее число значений: ничего, одно значение (ячейка), столбец значений (множество), таблицу значений (несколько столбцов):

1. При возврате одного значения обычно используются операторы сравнения (желательно '=' заменять на 'IN', в подзапросах желательно использовать обобщающие функции, которые всегда возвращают одно значение). Например,

```
SELECT * FROM t1
WHERE number > (SELECT AVG(rating) FROM t2);
```

2. При возврате множества значений (одного столбца) используется сравнение на принадлежность к множеству ('IN'); а также операторы ANY и ALL, которые используются совместно с операторами сравнения (ANY(SOME) – условие верно, если хоть одно значение, которое вернул подзапрос, удовлетворяет условию; ALL – условие верно, если все значения, которые вернул подзапрос, удовлетворяют условию). Например,

```
SELECT * FROM t1
WHERE number IN (SELECT number FROM t2);

SELECT * FROM t1
WHERE number <= ANY (SELECT number FROM t2);
```

```
SELECT * FROM t1
WHERE number > ALL (SELECT number FROM t2);
```

3. При возврате подзапросом таблицы (множество столбцов) можно

проверить только факт наличия данных с помощью оператора EXISTS (если подзапрос ничего не возвращает, то результат - ложь). Например,

```
SELECT *  
FROM t1  
WHERE EXISTS (SELECT * FROM t2);
```

Нет смысла использовать EXISTS, если подзапрос построен с помощью обобщающей функции, которая всегда возвращает значение.

Обычно внешний и внутренний подзапросы ничем не связаны, однако есть случаи, когда подзапрос должен использовать данные из внешнего запроса, такие подзапросы называются соотнесенными. Например, найти данные о фирмах, которые имеют филиалы в городе 'Minsk' (t1 {firm, boss, about}; t2 {firm, city, branch_address})

```
SELECT *  
FROM t1 A  
WHERE 'Minsk' IN (SELECT city FROM t2 B WHERE A.firm = B.firm);
```

Данный запрос работает следующим образом:

- 1) берется строка для внешнего запроса (строка-кандидат);
- 2) подзапрос выполняется, используя данные из строки-кандидата;
- 3) производится оценка условия для внешнего запроса
- 4) переход к следующей строке.

2.1.1.6. Многотабличные запросы

Для выборки значений из нескольких таблиц БД используются многотабличные запросы. В многотабличных запросах используются операции соединения таблиц (в основе которых лежит операция декартова произведения).

Процедура выполнения многотабличного запроса состоит в следующем:

- 1) выполняется секция FROM (формируется декартово произведение

таблиц и выполняется соответствующее соединение);

- 2) выполняется секция WHERE;
- 3) выполняется секция GROUP BY ... HAVING;
- 4) выполняется секция SELECT (формируются результирующие строки);
- 5) выполняется секция ORDER BY.

Виды соединений:

- 1) Декартово произведение двух таблиц: `SELECT t1.*, t2.* FROM t1, t2;`
- 2) Тета-соединение таблиц (используются знаки сравнения, на практике используется редко, так как трудно найти смысл)

`SELECT * FROM t1, t2 WHERE t1.number > t2.number;`

- 3) Экви-соединение таблиц (по равенству значений общего атрибута, например значений первичного и внешнего ключа):

`SELECT t1.*, t2.* FROM t1, t2 WHERE t1.number = t2.number;`

или эквивалентный вариант соединения (inner или natural join)

`SELECT t1.*, t2.* FROM t1 INNER JOIN t2 ON (t1.number = t2.number);`

- 4) Внешние соединения таблиц (левое, правое и полное):

`SELECT t1.*, t2.* FROM t1 LEFT JOIN t2 ON t1.number = t2.number;`

`SELECT t1.*, t2.* FROM t1 RIGHT JOIN t2 ON t1.number = t2.number;`

`SELECT t1.*, t2.* FROM t1 FULL JOIN t2 ON t1.number = t2.number;`

Кроме приведенных выше соединений, есть различные дополнительные применения многотабличных запросов, например (найти все пары студентов имеющих один и тот же рейтинг):

`SELECT A.name, B.name, A.rating FROM t1 A, t1 B WHERE A.rating = B.rating;`

Здесь псевдоним существует только на время выполнения SELECT.

2.1.2. Операторы изменения содержимого БД

2.1.2.1. Добавление (вставка) новых данных в таблицу

Новые данные можно добавлять только в одну таблицу. Варианты оператора:

1) вставка одной строки в таблицу:

```
INSERT INTO table [(column_list)]
```

```
VALUES (data_value_list);
```

где table - имя таблицы;

column_list - список имен столбцов (через запятую); если список не указан, то используется список из описания table; если не все столбцы перечислены, то в них подставляются NULL-значения или значения по умолчанию;

data_value_list - данные для вставки (типы соответствуют позициям column_list).

2) вставка множества строк (строки возвращает запрос, запрос должен возвращать столбцы, совпадающие по порядку со столбцами таблицы):

```
INSERT INTO table [(column_list)]
```

```
SELECT ...
```

2.1.2.2. Модификация данных в таблице

```
UPDATE table
```

```
SET column1 = value1 [, column2 = value2, ...]
```

```
[WHERE condition]
```

В SET указываются имена столбцов для изменения данных. Если есть секция WHERE, то будут изменены только те строки, для которых condition выполняется.

С UPDATE в WHERE можно использовать подзапросы (в том числе и те которые используют строки-кандидаты модифицируемой таблицы), однако нельзя в секции FROM подзапроса указывать модифицируемую таблицу.

2.1.2.3. Удаление данных из таблицы

DELETE FROM table

[WHERE condition]

Если есть секция WHERE, то будут изменены только те строки, для которых condition выполняется.

Как и в UPDATE, с DELETE в WHERE можно использовать подзапросы (в том числе и те которые используют строки-кандидаты модифицируемой таблицы), однако нельзя в секции FROM подзапроса указывать модифицируемую таблицу.

3. Порядок выполнения работы.

1. Проанализировать полученное от преподавателя задание – список запросов на естественном языке.

2. Построить запросы на языке SQL (в MS Access использовать для этого конструктор запросов в SQL-режиме), реализующие запросы задания. При этом допускается один исходный естественный запрос разбивать на несколько отдельных SQL-запросов (например, в MS Access запрос, имеющий имя «Запрос 1», может быть использован в запросе «Запрос 2» (работа с запросом будет описываться синтаксически как обращение к базовой таблице с именем «Запрос 1»), вызов запроса «Запрос 1» будет сформировано СУБД динамически), однако выполнение всей цепочки запросов должно задаваться в одном главном запросе.

3. Сформировать в таблицах БД необходимые данные для выполнения каждого запроса (необходимо учесть, что каждый запрос обязательно должен иметь не пустой результат).

4. Оформить отчет, включающий исходное задание и текст запросов на SQL. Выполнение запросов продемонстрировать исполнением запросов в СУБД и проверкой соответствия результатов запросов исходным данным.

ЛИТЕРАТУРА

- 1 Дейт К.Дж. Введение в системы баз данных, 6-е издание. – К.; М.; СПб.: ИД «Вильямс», 2000. с. 81-132, 269-350.
- 2 Грабер М. Справочное руководство по SQL. — М.: Лори, 1997. — 291 с.
- 3 Канолли Т., Бегг К., Страчан А. Базы данных: проектирование, реализация и сопровождение. 2-е издание. - К.; М.; СПб.: ИД «Вильямс», 2000. – с. 106-120.
- 4 Кириллов В.В. Основы проектирования реляционных баз данных - <http://citforum.ru/database/dbguide/index.shtml>.
- 5 Кузнецов С.Д. Основы современных баз данных - <http://citforum.ru/database/osbd/contents.shtml>.
- 6 Пушкинов А.Ю. Введение в системы управления базами данных - <http://citforum.ru/database/dblearn/index.shtml>.