Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра электронных вычислительных машин

Е. В. Калабухов

БАЗЫ ДАННЫХ, ЗНАНИЙ И ЭКСПЕРТНЫЕ СИСТЕМЫ

Лабораторный практикум
для студентов специальности I-40 02 01
«Вычислительные машины, системы и сети»
всех форм обучения

Минск 2008

УДК 681.3.016 (076)
ББК 32.973.26-018.2 я7
К 17

Р е ц е н з е н т
ведущий научный сотрудник
лаборатории №222 ОИПИ НАН Беларуси,
канд. техн. наук А. А. Дудкин

Калабухов, Е. В.

К 17 Базы данных, знаний и экспертные системы : лаб. практикум для студ. спец. I-40 02 01 «Вычислительные машины, системы и сети» всех форм обуч. / Е. В. Калабухов. – Минск : БГУИР, 2008. – 32 с. : ил.

ISBN 978-985-488-332-8

Лабораторный практикум посвящен вопросам проектирования реляционных баз данных. Рассмотрены теоретические и практические вопросы проектирования базы данных на всех стадиях обобщенной методологии проектирования. Приведены примеры работы с данными на языке SQL.

Практикум предназначен для студентов специальности I-40 02 01 «Вычислительные машины, системы и сети» всех форм обучения.

УДК 681.3.016 (076)
ББК 32.973.26-018.2 я7

ISBN 978-985-488-332-8

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель настоящего практикума – изучение *общей методологии проектирования реляционных баз данных*, а также изучение языка DML SQL для выполнения запросов к спроектированной базе данных (БД).

В рамках общей методологии проектирования практикумом охватываются следующие основные задачи:

1) *концептуальное проектирование БД* на основе построения ER-модели (ER-диаграммы) некоторого фрагмента реального мира;

2) *логическое проектирование БД* на основе построения предварительной реляционной модели данных на основании ER-диаграммы, а также выполнение нормализации предварительной реляционной модели данных и получение уточненной реляционной модели данных;

3) *физическое проектирование БД* – реализация уточненной реляционной модели данных в среде целевой СУБД.

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

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

Следует заметить, что данное пособие не рассматривается автором как справочник по какой-либо конкретной СУБД, поэтому выбор целевой СУБД возможен по желанию студента (исходя из возможностей учебной лаборатории) и в основном ограничен только рамками *реляционных* СУБД.

ЛАБОРАТОРНАЯ РАБОТА №1

СОЗДАНИЕ ER-МОДЕЛИ ДАННЫХ

1. Цель работы

Ознакомиться с задачами этапа *концептуального проектирования* БД. Изучить *ER-модель* представления данных (модель «сущность-связь»). Для указанного варианта задания разработать ER-модель данных с учетом семантических ограничений предметной области. Представить модель в виде *ER-диаграммы*.

2. Теоретические сведения

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

Цель построения модели данных – представление данных пользователя в понятном виде, который можно легко применить при проектировании БД. Модель данных 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.

Цель этапа *концептуального проектирования* БД – адекватное отображение предметной области и информационных потребностей пользователей в концептуальной модели данных.

Модель «сущность-связь» (*ER-модель*) представляет собой высокоуровневую концептуальную модель данных с возможностью графического представления информации в виде *ER-диаграмм*.

2.1. Концепции ER-модели

Можно выделить три основные семантические концепции в ER-модели:

1. *Объекты* (типы сущностей).

Типы объектов (типы сущностей) – множество объектов реального мира с одинаковыми свойствами, которые 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»). Каждый тип объекта идентифицируется уникальным именем и обязательным списком свойств.

Объект (*экземпляр* типа объекта или *сущность*) – экземпляр типа сущности, предмет, который может быть четко идентифицирован на основе

свойств (так как обладает уникальным набором свойств среди всех объектов одного типа).

Типы объектов классифицируются как *сильные* и *слабые*:

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

Представление объектов на диаграмме: в виде *прямоугольника* с именем внутри него; сильный тип объекта – прямоугольник с одинарным контуром; слабый тип объекта – прямоугольник с двойным контуром.

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 (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»);
- *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, «стаж сотрудника»; базовое – не зависит от других свойств;
- *ключевые и неключевые:* ключ – свойство (набор свойств), которое однозначно выделяет объект из всех объектов данного типа (например «номер паспорта»).

Представление свойств на диаграмме: в виде *эллипса* с уникальным именем (уникальность среди множества атрибутов) 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.

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 используется), 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. Число направлений связи (приложения функциональности связи к объектам) зависит от числа экземпляров объектов, участвующих в связи, и *всегда больше единицы*, т.е. любая связь не является однонаправленной.

Каждый тип отношения, как и тип объекта, идентифицируется обязательным именем, отражающим функции данной связи, и необязательным (но возможным) списком свойств.

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

Следует отметить, что приведенное выше описание концепций не является жестким и позволяет трактовать различные части реального мира как одну из концепций в зависимости от значимости данной части для создаваемой концептуальной модели. Например, такое понятие реального мира, как «семья», можно представить в виде объекта, а можно и как отношение между другими (более сильными) объектами («муж», «жена» и т.п.), если они уже существуют в создаваемой модели. Поэтому создание концептуальной модели данных более *искусство*, чем механическая работа, и требует формирования определенного (внутреннего) представления ситуации у разработчика.

2.2. Структурные ограничения ER-модели

Структурные ограничения, накладываемые на участников отношения, являются отражением требований реального мира. Можно выделить такие общие ограничения, как *мощность отношения* и *степень участия* объектов в отношении.

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 (обозначается «1»);
- *«один-ко-многим»* – максимальная мощность отношения в одном направлении равна одному, а в другом – многим (обозначается «*»);
- *«многие-ко-многим»* – максимальная мощность отношения в обоих направлениях равна 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 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.

Пример ER-диаграммы приведен на рис. 1, а рекомендации по ее созданию приведены ниже.

3. Рекомендации по выполнению работы

Рекомендуется следующий *порядок выполнения и оформления работы*:

Этап 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 или 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.)).

В качестве рабочего примера по созданию ER-диаграммы рассмотрим направление *«Университет»*. В модели университета нашим основным интересом будет описание учебного процесса – состав учебных групп, оценки студентов, расписание занятий (остальная часть реального университета нас не интересует и не будет реализована в рамках концептуальной модели).

Этап 2. Сформировать *объекты* (для учебной модели требуется не менее 5–6 объектов сильного типа). Для этого рекомендуется:

1. Выделить *единичные объекты* предметной области (естественно, не все, но так, чтобы «ассортимент» различных объектов был как можно шире). Этот этап необходим для более полного осмысления предметной области и также может быть выполнен умозрительно. Например, в модели «университет» можно представить себе такие единичные объекты, как «студент Иванов», «группа 050505», «предмет БДЗиЭС» и т.п. Подобные действия можно выполнить также с помощью интервью, проводимого с заказчиком (представителем предметной области).

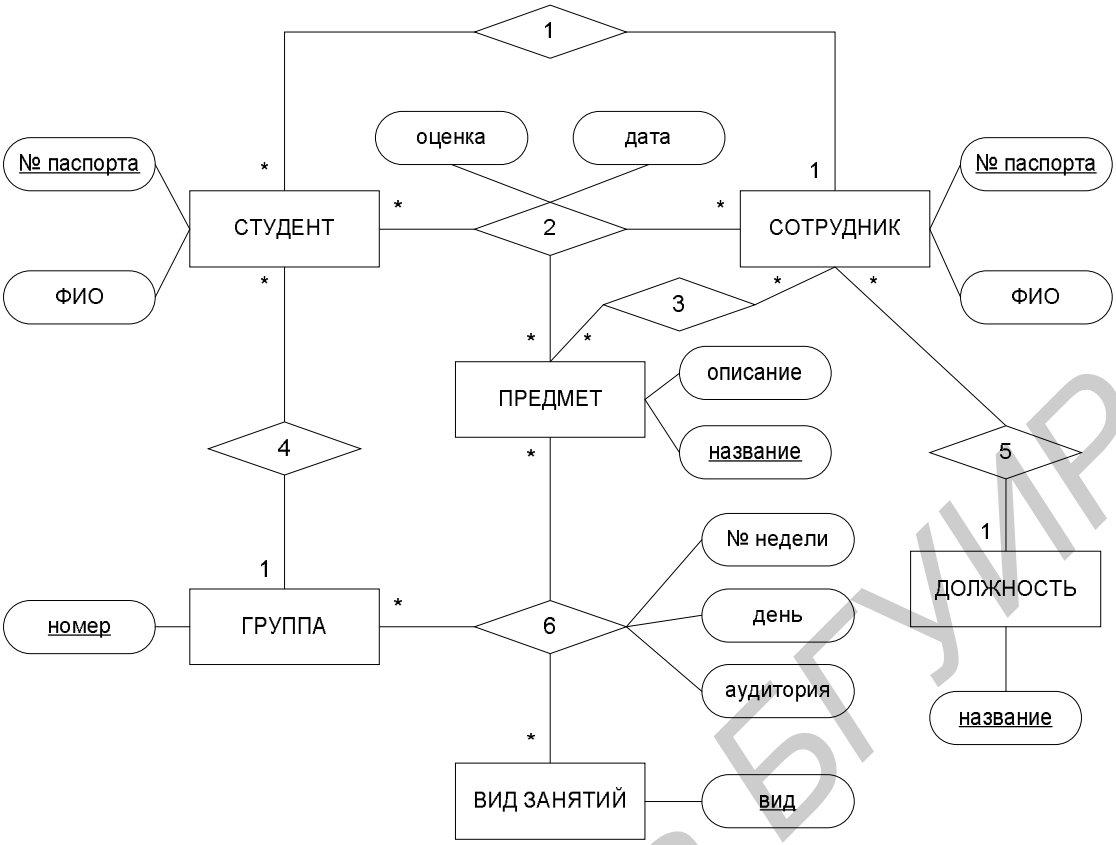


Рис. 1. ER-диаграмма «Университет»

2. Выделить *классы* объектов (множества качественно (атрибутивно) сходных объектов). Для модели «университет» можно представить себе такие классы объектов, как «студент», «группа», «предмет» и т.п. (см. рис. 1). Следует отметить, что в рамках общей методологии проектирования (в связи с усложнением диаграммы и дальнейших операций над данными, а также низкой информативностью) *не рекомендуется*:

- выделять в отдельные классы такие объекты, которые всегда (за время существования БД) будут присутствовать в только единичном экземпляре; для нашего примера таким объектом можно назвать объект «университет», т.к. он всегда будет содержать только один экземпляр – «БГУИР»;
- формировать классы объектов, различающиеся по одному или нескольким признакам при прочих одинаковых атрибутах и общей семантической направленности объектов; например, делить преподавателей на классы «преподаватель физики», «преподаватель математики» и т.п.;
- создавать иерархии объектов путем выделения подмножеств экземпляров объектов с одинаковыми свойствами; например, «сотрудники» – «преподаватели» – «преподаватели кафедры»; дублирование информации также недопустимо с точки зрения возможного нарушения целостности БД;

- выделять в отдельные классы объекты, *сильно зависимые* от других классов (обычно двух и более), т.е. более выражающие связь между объектами.

3. Перевести полученные классы *в объекты* на ER-диаграмме, причем у каждого объекта желательно выделить не менее двух атрибутов и обязательно выбрать из них *ключевое(ые)*. Значения атрибутов должны быть определены для всех экземпляров объектов. *Не рекомендуется* создавать атрибуты, имеющие зависимость от числа других объектов (например «оценка по физике», «оценка по математике»), т.к. это снижает надежность структуры БД и ведет к сложной обработке данных, в таких случаях лучше работает связь с другим объектом. Также, если один объект имеет атрибут, семантически схожий с другим объектом модели, то данный атрибут следует заменить (исходя из соображений целостности БД) на связь между указанными объектами. В учебной модели желательно исключить атрибуты, хранящие статистическую информацию (например «рейтинг студента»), особенно если эти значения можно будет рассчитать другим путем.

Этап 3. Сформировать *связи*. Для этого рекомендуется:

1. Оценить, как могут быть взаимосвязаны между собой экземпляры объектов разных типов объектов. Причем в рамках учебной БД необходимо сформировать не только связи иерархии (например «студент» – «группа»), но и связи *производственных отношений* (например «студент» – «предмет»), которые должны содержать дополнительные атрибуты и могут отражать достаточно мелкие события (например оценки студентов по предметам). При формировании связей желательно учесть следующее:

- между объектами могут быть заданы разные связи, по одной для каждой функциональности;
- схожие по функциональности связи между одинаковыми участниками могут быть объединены в одну с добавлением соответствующих атрибутов связи;
- при наличии более сложной связи (по числу участников) можно исключать похожие (дублирующие) связи меньших степеней (для одних и тех же участников и для одной и той же функциональности);
- атрибуты связи, которые семантически схожи с некоторым объектом модели, следует удалить и внести этот объект в качестве дополнительного участника связи;
- необходимо добавить такие дополнительные атрибуты связи, которые будут расширять возможные комбинации отношений объектов (например «дата экзамена»).

Для модели университета можно выделить следующие связи (с учетом их функциональной направленности и выделенных типов объектов):

1 – «студент – декан», описывает подчиненность студента декану (здесь декан – это подмножество из числа сотрудников; возможность установки связи может быть описана как дополнительное бизнес-правило: «сотрудник должен иметь должность декан»);

2 – «журнал оценок», описывает оценки всех студентов по всем предметам, выставленные преподавателями (здесь преподаватель – подмножество от числа сотрудников; возможность выставления оценки может быть описана как дополнительное бизнес-правило: «сотрудник имеет должность из списка преподавателей»);

3 – «знание предмета», описывает предметы, занятия по которым может вести преподаватель;

4 – «состав группы», описывает вхождение студентов в учебные группы;

5 – «должность сотрудника», описывает должность, которую занимает сотрудник;

6 – «расписание предметов», описывает проведение занятий по аудиториям.

2. Установить на связи *структурные ограничения* с учетом текущей ситуации, а также возможных расширений БД при ее эксплуатации. Учесть, что в рамках общей методологии проектирования:

- связи мощности «один-к-одному» возникают в моделях крайне редко, поэтому лучше заменить такую связь связью мощности «один-ко-многим»;
- связи с числом участников более двух лучше задавать как связь мощности «много» со стороны всех участников (дополнительные ограничения можно пояснить на основе выработанных бизнес-правил и учесть на следующих этапах проектирования).

Этап 4. Полученную концептуальную диаграмму еще раз *проверить* по замечаниям *этапов* 1 – 3, т.к. процесс формирования концептуальной модели в рамках общей методологии проектирования БД носит *итерационный* характер.

Этап 5. Оформить *отчет*, включающий в себя исходное задание и конечную концептуальную ER-диаграмму (допустимо указывать дополнительные пояснения, если семантика фрагментов диаграммы не ясна из названий).

4. Контрольные вопросы

1. Понятие модели данных.
2. Цель этапа концептуального проектирования БД.
3. Концепции ER-модели.
4. Структурные ограничения ER-модели.
5. Обозначения концепций и структурных ограничений на ER-диаграмме.
6. Основные этапы формирования концептуальной модели данных.
7. Сравнение ER-модели и объектно-ориентированной модели.

ЛАБОРАТОРНАЯ РАБОТА №2

СОЗДАНИЕ РЕЛЯЦИОННОЙ МОДЕЛИ ДАННЫХ ПО ER-МОДЕЛИ

1. Цель работы

Познакомиться с *реляционной моделью данных*. Для указанного варианта задания преобразовать ER-диаграмму в реляционную модель данных. Реализовать полученную реляционную модель данных в среде целевой СУБД.

2. Теоретические сведения

Реляционная модель данных – формальная теория данных, основанная на некоторых положениях математики (теории множеств и предикативной логике).

2.1. 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).

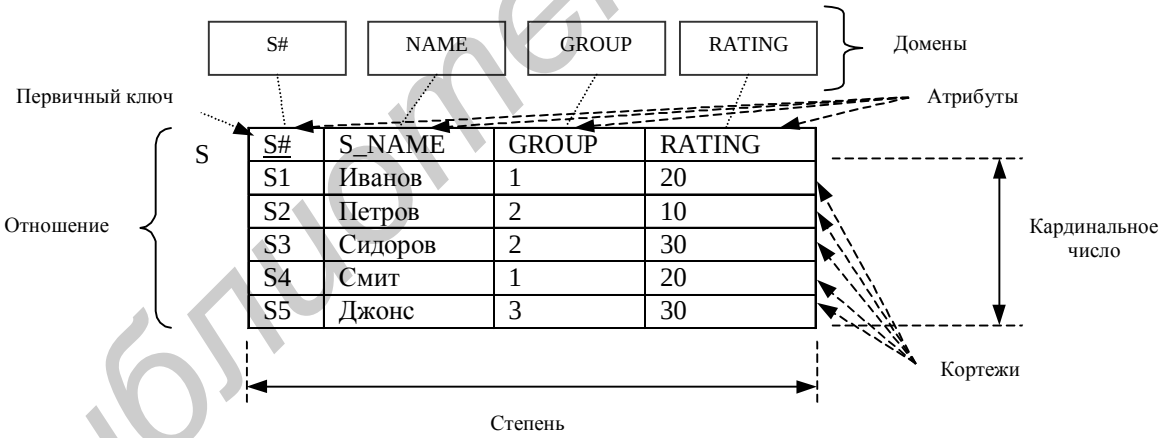


Рис. 2. Реляционные объекты данных (отношение S)

Краткое описание терминов реляционной модели (рис. 2):

- 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.

Отношение *R*, определенное на множестве доменов *D1, D2, …, Dn* (необязательно различных), содержит две части: *заголовок* (строка заголовков столбцов в таблице) и *тело* (строки таблицы).

Заголовок содержит фиксированное множество атрибутов или пар вида <имя_атрибута : имя_домена>: *{<A1:D1>, <A2:D2>, …, <An:Dn>}*, причем каждый атрибут *Aj* соответствует одному и только одному из лежащих в основе доменов *Dj (j=1,2,…,n)*. Все имена атрибутов *A1, A2, …, An* – разные.

Тело содержит множество кортежей. Каждый кортеж, в свою очередь, содержит множество пар <имя_атрибута : значение _атрибута>:

{<A1:vi1>, <A2:vi2>, …, <An:vin>},

(*i=1,2,…,m*, где *m* – количество кортежей в этом множестве). В каждом таком кортеже есть одна пара <имя_атрибута : значение_атрибута>, т.е. *<Aj:vij>*, для каждого атрибута *Aj* в заголовке. Для любой такой пары *<Aj:vij> vij* является значением из уникального домена *Dj*, который связан с атрибутом *Aj*.

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; важное следствие из этого свойства – поскольку отношение не содержит одинаковых кортежей, то всегда существует первичный ключ отношения;
- в любом отношении кортежи не упорядочены; следствие – нет понятий позиции кортежа и последовательности кортежей в отношении, кортеж в отношении всегда определяется по первичному ключу;
- в любом отношении атрибуты не упорядочены; следствие – атрибут всегда определяется по имени, нет понятий позиции атрибута и последовательности атрибутов в отношении;
- в отношении все значения атрибутов атомарные (неделимые); следствие – в каждой ячейке на пересечении столбца и строки в таблице расположено только одно значение (что 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 отношение.

2.2. Целостность реляционных данных

Целостность данных предназначена для сохранения в БД «отражения действительности реального мира», т.е. устранения недопустимых конфигураций (состояний) значений и связей, которые не имеют смысла в реальном мире.

Правила целостности данных можно разделить на:

- специфические или корпоративные ограничения целостности –дополнительные ограничения, специфические для конкретных БД;

- общие правила целостности – правила, которые применимы к любой реляционной БД (относятся к потенциальным (первичным) и к внешним ключам):

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* с одинаковым значением *K*;
- свойством неизбыточности (неприводимости) – никакое из подмножеств *K* не обладает свойством уникальности.

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 (т.е. 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 один из потенциальных ключей должен быть выбран в качестве первичного ключа отношения, а остальные потенциальные ключи будут называться альтернативными ключами.

2. *Внешние ключи.*

Пусть *R2* – базовое отношение. Тогда внешний ключ *FK* в отношении *R2* – это подмножество множества атрибутов *R2*, такое, что:

- существует базовое отношение *R1* (при этом *R1* и *R2* необязательно различны) с потенциальным ключом *K*;
- каждое значение *FK* в текущем значении *R2* или является *NULL*-значением, или совпадает со значением *K* некоторого кортежа в текущем значении *R1*.

Внешний ключ будет составным (т.е. будет состоять более чем из 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.

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.

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 B ссылается на A, тогда A 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;
- при попытке обновить потенциальный ключ, на который ссылается внешний ключ.

4. *NULL-значения.*

NULL-значения (определитель *NULL*) введены для обозначения таких значений атрибутов, которые на настоящий момент неизвестны или неприемлемы для некоторого кортежа. Это не значение по умолчанию, а отсутствие какого-либо значения (например данные об адресе нового студента (на данный момент)).

Для каждого атрибута должно быть установлено, может ли он принимать *NULL*-значения или нет, т.к. это влияет на концепции потенциальных и внешних ключей реляционной модели данных:

- *целостность объектов* – в реляционной модели данных ни один атрибут потенциального ключа 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»).

2.3. Получение реляционной модели из ER-диаграммы

Алгоритм преобразования ER-диаграммы в реляционную модель (схему) состоит из следующих шагов:

Шаг 1. Каждый *объект* на ER-диаграмме превращается в таблицу. Имя объекта становится именем таблицы.

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

Шаг 3. Уникальные (*ключевые*) атрибуты объекта превращаются в первичный ключ таблицы. Если имеется несколько возможных уникальных идентификаторов, то выбирается наиболее подходящий для использования.

Шаг 4. Связи *«один-ко-многим»* (в том числе и связи *«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.

Шаг 5. Связи *«многие-ко-многим»* реализуются через промежуточную таблицу. Эта таблица будет содержать как минимум столбцы внешних ключей на соответствующие объекты. Первичный ключ промежуточной таблицы должен включать в себя все внешние ключи на объекты, участвующие в связи.

Шаг 6. Если связь имеет *дополнительные* атрибуты, то, как и в случае атрибутов объектов, они становятся возможным столбцом таблицы:

- для связей «один-ко-многим» – в таблице со стороны «многие» (вместе с внешним ключом);
- для связей «многие-ко-многим» – в промежуточной таблице (при этом атрибуты, расширяющие комбинацию в связи (например «дата»), также должны войти в состав первичного ключа промежуточной таблицы).

3. Рекомендации по выполнению работы

Рекомендуется следующий *порядок выполнения и оформления работы*:

Этап 1. Проанализировать исходную ER-диаграмму (получена в лабораторной работе №1 или выдана преподавателем) и получить реляционную модель данных для этой диаграммы. Для этого рекомендуется:

1. Используя приведенный в п. 2.3 алгоритм, представить полученную реляционную модель графически в виде схемы данных. При этом не забыть рассмотреть *типы данных* (фактически аналог доменов) для столбцов (это особенно важно для ключевых полей).

2. Реализовать полученные реляционные отношения в виде таблиц в среде целевой СУБД. При этом рекомендуется имена таблиц и столбцов таблиц задавать латиницей (без наличия в именах пробелов). Имена также должны иметь семантический смысл.

3. Задать первичные ключи таблиц (это может потребовать коррекции свойств ключевых полей).

4. Задать внешние ключи таблиц (ссылка должна указывать на *первичный ключ* целевой таблицы).

5. Установить свойства целостности данных (каскадное удаление и обновление), если это возможно в целевой СУБД.

6. Проверить полученную реляционную модель путем внесения в таблицы строк данных (1-2 строки на таблицу, сначала заполняются целевые таблицы).

Для рассмотренной в первой лабораторной работе концептуальной модели «Университет» реляционная схема данных приведена на рис. 3.

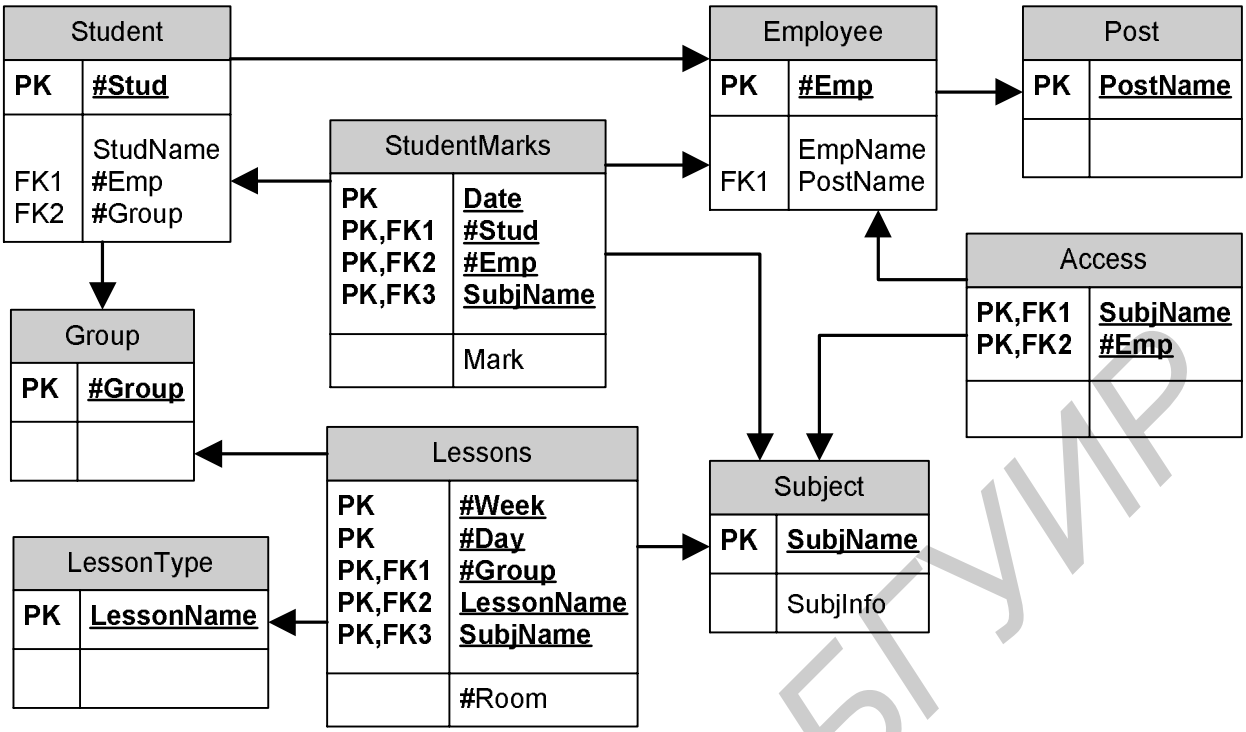


Рис. 3. Реляционная схема данных «Университет»

Этап 2. Оформить отчет, включающий исходную ER-диаграмму и конечную схему реляционной БД.

4. Контрольные вопросы

1. Понятие отношения в реляционной модели.
2. Понятие целостности реляционных данных.
3. Понятие потенциального и первичного ключа.
4. Понятие ссылки (внешнего ключа) и ссылочной целостности.
5. Представление объектов ER-модели в реляционной модели.
6. Представление связей ER- модели «один-ко-многим» в реляционной модели.
7. Представление связей ER-модели «многие-ко-многим» в реляционной модели.

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. Теоретические сведения

Нормализация – метод создания набора отношений с заданными свойствами на основе некоторых требований к данным. 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:

- *аномалии вставки* – возникают при добавлении новых несогласованных данных (нарушающих целостность данных в отношении);
- *аномалии изменения* – возникают при изменении части ранее введенных данных; частичное обновление сведений приведет к нарушению целостности данных отношения;
- *аномалии удаления* – возникают при удалении строк из отношений.

Обычно для решения проблем избыточности и аномалий выполняется деление 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 A (A является детерминантом атрибута B)

$$A \to B,$$

то в каждом кортеже этого отношения каждое конкретное значение атрибута A всегда связано только с одним значением атрибута B. Атрибуты A и B 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 понять, 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.

2.3. Нормальные формы и нормализация методом декомпозиции

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

Суть процесса нормализации:

- в нормализованных отношениях не разрешаются никакие функциональные зависимости, кроме функциональных зависимостей вида $K \to A$, где K – потенциальный ключ отношения R, а A – неключевой атрибут;
- если же отношение R имеет функциональные зависимости $B \to A$, где B не является потенциальным ключом, то в отношении R будет наблюдаться избыточность данных.

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:

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

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), т.е. всегда можно использовать его результат для обратного преобразования, т.к. в процессе нормализации не утрачиваются первоначальные функциональные зависимости.

Используемые в лабораторной работе нормальные формы:

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.

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 $A \to B$, 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 $A \to C$, если существуют зависимости $A \to B$ и $B \to C$ (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 *ЗНФ* эквивалентны. В *НФБК* устраняются аномалии, связанные с функциональными зависимостями не от потенциальных ключей отношения.

Декомпозиция – 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 с использованием соединения кортежей отношений, полученных в результате декомпозиции;
- сохранение зависимостей – функциональные зависимости при декомпозиции сохраняются.

Для выполнения процесса декомпозиции вначале 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 *ЗНФ*;
- 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.

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. Рекомендации по выполнению работы

Рекомендуется следующий *порядок выполнения и оформления работы*:

Этап 1. Выделить функциональные зависимости для каждого отношения исходной реляционной схемы (полученной на лабораторной работе №2 или выданной преподавателем). Проверить практический смысл выделенных функциональных зависимостей.

Этап 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 *ЗНФ* (для каждого отношения должен быть задан первичный ключ).

Этап 3. Для полученной нормализованной реляционной схемы проверить смысл ссылок.

Этап 4. Реализовать полученные реляционные отношения в виде таблиц в среде целевой СУБД.

Этап 5. Оформить отчет, включающий исходную и нормализованную схему БД.

4. Контрольные вопросы

1. Аномалии реляционных схем данных.
2. Понятие функциональной зависимости.
3. Суть процесса нормализации.
4. Правила нормальных форм 1НФ и 2НФ.
5. Правила нормальных форм 3НФ и НФБК.
6. Процесс нормализации методом декомпозиции.

ЛАБОРАТОРНАЯ РАБОТА №4

РАБОТА С РЕЛЯЦИОННОЙ БАЗОЙ ДАННЫХ НА ЯЗЫКЕ SQL

1. Цель работы

Познакомиться с операторами языка *SQL*, отвечающими за выборку, добавление, модификацию и удаление данных из БД. Для выданного варианта задания представить запросы на языке SQL, реализованные в среде целевой СУБД.

2. 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

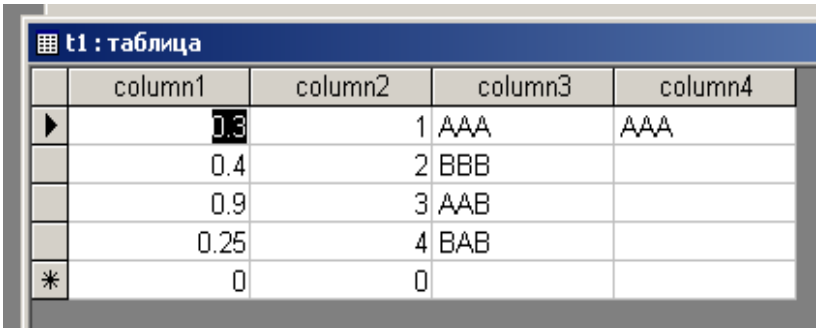
```
SELECT [DISTINCT | ALL] {* | [column [AS new_column_name]] [,…]}
FROM table [alias] [,…]
[WHERE condition]
[GROUP BY list [HAVING condition]]
[ORDER BY list]
```

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 (см. выполнение на рис. 4):

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



а

б

Рис. 4. Выполнение примера 1:
а – исходные данные; б – результат выполнения

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 (см. результат на рис. 5):

```
SELECT *
FROM t1
WHERE (column1>0.2) And (column2 IN (1,2,3)) And (column3 LIKE '_A%')
AND (column4 IS NULL);
```

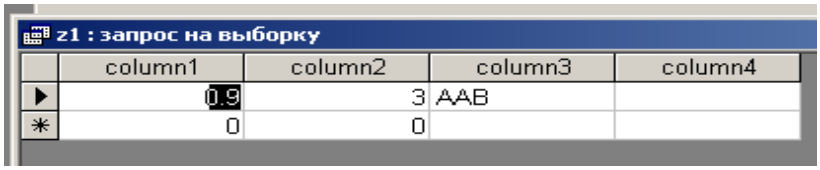


Рис. 5. Результат выполнения примера 2 (см. исходные данные на рис. 4)

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 (см. результат на рис. 6):

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

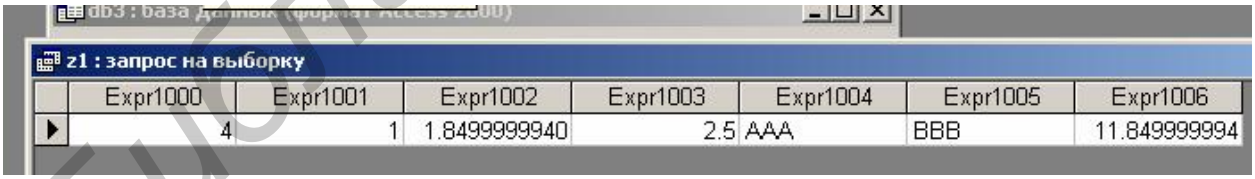


Рис. 6. Результат выполнения примера 3 (см. исходные данные на рис. 4)

2.1.1.4. Группирующие запросы (секция *GROUP BY*)

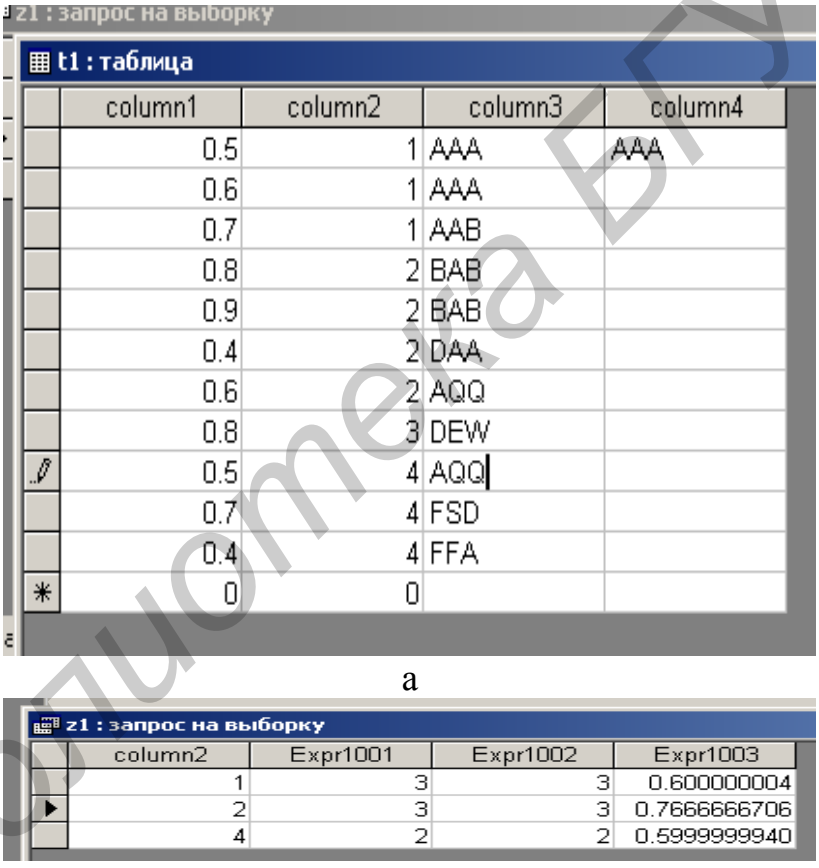
Группирующие запросы служат для обобщения (группировки) результатов по одинаковым значениям (группам) в указанных после *GROUP BY* столбцах. Для каждой группы в таком запросе создается только одна строка результата. Все имена столбцов, приведенные в описании *SELECT*, должны

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 (см. выполнение на рис. 7):

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



а

б

Рис. 7. Выполнение примера 4:
а – исходные данные; б – результат выполнения

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 (см. выполнение на рис. 8):

```
SELECT *
FROM t1
ORDER BY column1, column2 DESC, column3;
```

t1 : таблица

	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

z1 : запрос на выборку

	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

Рис. 8. Выполнение примера 5:
а – исходные данные; б – результат выполнения

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 располагаться 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. При использовании *ANY* (*SOME*) – условие верно, если хоть одно значение, которое вернул подзапрос, удовлетворяет условию. При использовании *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 – ложен). 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’ (используемые таблицы: *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 (строка-кандидат) из таблицы *t1* для внешнего запроса;

2) подзапрос выполняется с использованием данных из текущей строки-кандидата (т.е. значения *A.firm*);

3) производится оценка условия для внешнего запроса (принятие решения для формирования выходных данных);

4) выполняется переход к следующей необработанной строке таблицы *t1* и процесс повторяется.

2.1.1.7. 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 {>, >=, <, <=}, что на практике используется редко, так как трудно найти смысл такого сравнения):

*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 данных возвращаются запросом *SELECT*, при этом число столбцов возвращаемых запросом и их типы должны

соответствовать столбцам таблицы *table*):

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. Рекомендации по выполнению работы

Рекомендуется следующий *порядок выполнения и оформления работы*:

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

Этап 2. Построить запросы на языке *SQL*, реализующие запросы задания. При этом допускается один исходный естественный запрос разбивать на несколько отдельных *SQL*-запросов, однако выполнение всей цепочки запросов должно задаваться в одном главном запросе.

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

Этап 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.

4. Контрольные вопросы

1. Простой оператор SELECT.
2. Возможности секции WHERE.
3. Сортировка результатов.
4. Обобщающие функции и группирующие запросы.
5. Подзапросы.
6. Многотабличные запросы.
7. Операторы изменения содержимого БД.

ЛИТЕРАТУРА

1. Ульман, Д. Системы баз данных. Полный курс / Д. Ульман, Г. Гарсиа-Молина, Д. Уидом. – СПб. : ИД «Вильямс», 2004. – 1088 с.
2. Грабер, М. Введение в SQL / М. Грабер. – М. : Лори, 1996. – 320 с.
3. Дейт, К. Дж. Введение в системы баз данных / К. Дж. Дейт. – СПб. : ИД «Вильямс», 2006. – 1328 с.
4. Канолли, Т. Базы данных: проектирование, реализация и сопровождение. Теория и практика / Т. Канолли, К. Бегг, А. Страчан. – СПб. : ИД «Вильямс», 2003. – 1436 с.
5. Роб, П. Системы баз данных: проектирование, реализация и управление / П. Роб, К. Коронел. – СПб. : БХВ, 2004. – 1040 с.
6. Ульман, Д. Основы реляционных баз данных / Д. Ульман, Д. Уидом. – М. : Лори, 2006. – 382 с.
7. Харрингтон, Д. Л. Проектирование реляционных баз данных / Д. Л. Харрингтон. – М. : Лори, 2006. – 230 с.
8. Хомоненко, А. Д. Базы данных / А. Д. Хомоненко, В. М. Цыганков. – СПб. : Корона, 2000. – 355 с.

Св. план 2008, поз. 57

Учебное издание

Калабухов Евгений Валерьевич

БАЗЫ ДАННЫХ, ЗНАНИЙ И ЭКСПЕРТНЫЕ СИСТЕМЫ

Лабораторный практикум
для студентов специальности I-40 02 01
«Вычислительные машины, системы и сети»
всех форм обучения

Редактор Т. П. Андрейченко
Корректор Е. Н. Батурчик

Подписано в печать 10.07.2008. Формат 60×84 1/16. Бумага офсетная.
Гарнитура «Таймс». Печать ризографическая. Усл. печ. л. 2,09.
Уч.-изд. л. 2,0. Тираж 100 экз. Заказ 240.

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0056964 от 01.04.2004. ЛП №02330/0131666 от 30.04.2004.
220013, Минск, П. Бровки, 6