**Лабораторная работа №1. Проектирование базы данных.**

**Цель работы:**

– получить практические навыки создания концептуальной и логической моделей базы данных.

**Теоретические сведения**

## 

## Этапы проектирования базы данных

Полный цикл разработки базы данных включает концептуальное, логическое и физическое ее проектирование.

Основными целями проектирования базы данных являются:

* представление данных и связей между ними, необходимых для всех основных

областей применения данного приложения и любых существующих групп его пользователей;

* создание модели данных, способной поддерживать выполнение любых требуемых транзакций обработки данных;
* разработка предварительного варианта проекта, структура которого позволяет

удовлетворить требования, предъявляемые к производительности системы.

1. **Концептуальное проектирование**

На этапе концептуального проектирования определяются все сущности предметной области, их основные атрибуты и связи. Результатом концептуального проектирования является концептуальная модель базы данных.

**Фундаментальные понятия**

Главными элементами концептуальной модели базы данных являются *сущности, их атрибуты* и  *связи.* Проектирование базы данных начинается с построения модели предметной области. Наиболее распространенной моделью предметной области является ER*-диаграмма* (entity-relationship diagramm) – диаграмма «**Сущность-связь»**.

*Существуют различные системы графического изображения ER-диаграмм (нотации).*

**НотацияЧена.**

Таблица 1. Обозначения элементов базы данных в нотации Чена

|  |  |
| --- | --- |
| **Элемент ER-диаграммы** | **Обозначение** |
| Сотрудник Независимая сущность | Сотрудник |
| Зависимая сущность  Сотрудник |  |
| Атрибут сущности (свойство, характеризующее объект) |  |
| Ключевой атрибут (атрибут, входящий в первичный ключ) |  |
| Связь | **1:М** |

Сущность и ее атрибуты на ER-диаграмме соединяются ненаправленными дугами как показано на рис.1.

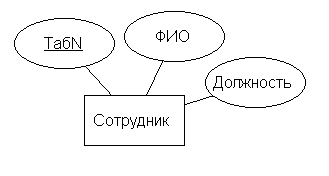


Рис.1. Объект и его атрибуты на ER-диаграмме

Между сущностями существуют связи, каждая из которых относится к одному из 3-х типов:

1. Связь типа**1:1 «Один – к одному»**. Этот тип связи означает, что каждому объекту первого вида соответствует не более одного объекта второго вида, и наоборот.

*Например*: сотрудник может руководить только одним отделом, и у каждого отдела есть только один руководитель. Фрагмент ER-диаграммы со связью «Один к одному» показан на рис.2.

Сотрудник

Отдел

1 : 1 11

Рис.2. Фрагмент ER- диаграммы со связью «Один к одному"

2. Связь типа **1:M «Один ко многим»*.***Этот тип связи означает, что каждому объекту первого вида может соответствовать более одного объекта второго вида, но каждому объекту второго вида соответствует не более одного объекта первого вида. Эта связь обозначается аналогично связи «Один к одному», только внутри ромба записывается текст «1 : М». *Например,* в каждом отделе может быть множество сотрудников, но каждый сотрудник работает только в одном отделе

* 1. Связь типа **N:M «Многие ко многим».** Этот тип связи означает, что каждому объекту первого вида может соответствовать более одного объекта второго вида, и наоборот. У этого типа связи иногда бывают собственные атрибуты.

*Например,*: каждый счет может включать множество товаров, и каждый товар может входить в разные счета.

**Нотация Мартина**

|  |  |
| --- | --- |
| **Элемент диаграммы** | **Обозначение** |
| независимая сущность | http://www.mstu.edu.ru/study/materials/zelenkov/image144.gif |
| зависимая сущность | http://www.mstu.edu.ru/study/materials/zelenkov/image41.gif |
| Связь | http://www.mstu.edu.ru/study/materials/zelenkov/image148.gif |

Список атрибутов приводится внутри прямоугольника, обозначающего сущность. Ключевые атрибуты подчеркиваются. Связи изображаются линиями, соединяющими сущности, вид линии в месте соединения с сущностью определяет кардинальность связи.

Под кардинальностью связи понимается пара чисел, определяющая модальность и мощность связи в месте соединения этой связи с сущностью.

Каждая связь в месте соединения с сущностью может иметь одну из двух модальностей: **«может»** и **«должен»**. Модальность **«может»** означает, что экземпляр одной сущности может быть связан с одним или несколькими экземплярами другой сущности, а может быть и не связан ни с одним экземпляром. Модальность «**должен»** означает, что экземпляр одной сущности обязан быть связан не менее чем с одним экземпляром другой сущности. Связь может иметь разную модальность с разных концов. Необязательность связи (**«может»**) обозначается пустым кружочком на конце связи или цифрой 0 над связью, а обязательность (**«должен»**) - перпендикулярной линий, перечеркивающей связь или цифрой 1 над связью.

Мощность связи определяет количество строк сущности (кортежей), участвующих в связи. Количество кортежей, участвующих в связи может быть «один» (1) или «много» (N) в соответствии с типом связи. Возможные значения кардинальности связи приведены в таблице 2.

Таблица 2 – Обозначения кардинальности связи в нотации Мартина

|  |  |  |
| --- | --- | --- |
| **Значение**  **(модальность, мощность)** | **Кардинальность** | **Обозначение** |
| Может / Один | 0,1 | http://unesco.kemsu.ru/study_work/method/DB/book/img/clip_image021.gif |
| Должен / Один | 1,1 | http://unesco.kemsu.ru/study_work/method/DB/book/img/clip_image020.gif |
| Может /Много | 0,N | http://unesco.kemsu.ru/study_work/method/DB/book/img/clip_image023.gif |
| Должен/Много | 1,N | http://unesco.kemsu.ru/study_work/method/DB/book/img/clip_image024.gif |

На рис. 3 представлен фрагмент ER-диаграммы в нотации Мартина.

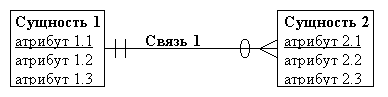


Рис. 3. Фрагмент ER-диаграммы в нотации Мартина

**Нотация Баркера**

Сущности обозначаются прямоугольниками, внутри которых приводится список атрибутов. Ключевые атрибуты отмечаются символом # (решетка). Связи обозначаются линиями с именами, место соединения связи и сущности определяет кардинальность связи.

Таблица 3. Обозначения кардинальность связи в нотации Баркера.

|  |  |
| --- | --- |
| **Обозначение** | **Кардинальность** |
| http://unesco.kemsu.ru/study_work/method/DB/book/img/clip_image027.gif | 0,1 |
| http://unesco.kemsu.ru/study_work/method/DB/book/img/clip_image019.gif | 1,1 |
| http://unesco.kemsu.ru/study_work/method/DB/book/img/clip_image035.gif | 0,N |
| http://unesco.kemsu.ru/study_work/method/DB/book/img/clip_image036.gif | 1,N |

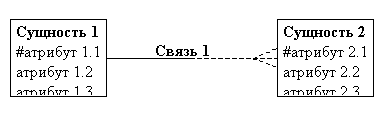


Рис. 14.6. Пример ER-диаграммы в нотации Баркера

### 2. Логическое проектирование БД

*Логическое проектирование* заключается в определении числа и структуры таблиц, включая названия столбцов, типы данных и ограничения в столбцах, первичные и внешние ключи, а также свойства связей.

## На этапе логического проектирования реляционной базы данных необходимо решить следующие задачи:

1. Преобразовать ER-диаграмму в схему БД.
2. Удалить избыточное дублирование данных.
3. Определить типы данных и ограничения для полей таблиц.
4. Определить первичные и внешние ключи в таблицах.
5. Описать правила обеспечения ссылочной целостности для связей.
6. Провести нормализацию таблиц

Готовая логическая модель базы данных представляет собой таблицы для каждой сущности концептуальной модели. В таблице должны быть указаны

* Имена всех полей
* Типы и размеры данных всех полей
* Доменные ограничения
* Правила обеспечения ссылочной целостности для связанных таблиц

### Преобразование ER-диаграммы в логическую модель базы данных

Правила преобразование ER-диаграммы в схему БД следующие:

1. Каждый тип сущности преобразуется в таблицу БД. В таблицу вносятся все атрибуты, относящиеся к данному типу сущности.
2. Cвязь 1:N между сущностями реализуется с помощью пары Первичный ключ – Внешний ключ между двумя таблицами.
3. Связь 1:1 может быть удалена, а связанные этой связью две таблицы могут быть объединены в одну таблицу.
4. Cвязь типа M:N реализуется с помощью промежуточной таблицы. Например, для сущностей *СТУДЕНТЫ* и *ПРЕПОДАВАТЕЛИ*, имеющих связь M:N, промежуточная таблица *ОЦЕНКИ* может содержать два внешних ключа: идентификатор студента и идентификатор преподавателя, а также идентификатор дисциплины и оценку, полученную студентом по этой дисциплине. В этом случае

вместо связи M:N между таблицами СТУДЕНТЫ и ПРЕПОДАВАТЕЛИ в базе данных будут присутствовать две связи 1:N между таблицей ОЦЕНКИ и таблицами СТУДЕНТЫ и ПРЕПОДАВАТЕЛИ.

### Определение первичных ключей

В принципе можно создать таблицу и без первичного ключа. Но при создании первичного ключа создается и кластеризованный индекс для таблицы, поэтому наличие у каждой таблицы первичного ключа – хороший стиль проектирования БД. Кроме того, если эта таблица является родительской для какой-либо другой таблицы, то определить первичный ключ необходимо, чтобы можно было определить внешний ключ в подчинённой таблице. В качестве первичного ключа следует выбирать уникальный атрибут сущности, по которому чаще всего происходит обращение к данным. Например, для БД налоговой инспекции это ИНН – индивидуальный номер налогоплательщика, а для БД ФОМС (фонда обязательного медицинского страхования) – номер медицинского полиса.

Если у сущности нет уникальных атрибутов, можно рассмотреть уникальные комбинации атрибутов. Но первичный ключ не должен быть длинным, т.к. ссылающийся на него внешний ключ будет занимать много памяти. Поэтому при отсутствии подходящих атрибутов нужно вводить суррогатный первичный ключ, который не несёт смысловой нагрузки и служит только для идентификации записей. Некоторые СУБД позволяют определять значения такого ключа как AUTOINCREMENT (Identity в SQL SERVER), т.е. числовое поле, значение которого начинается с 1 и автоматически увеличивается на 1 при добавлении новой записи. Хорошей практикой считается создание в каждой таблице суррогатного первичного ключа в виде целочисленного поля с автозаполнением.

### Определение типов данных полей таблиц

Определение типов данных для полей таблиц зависит от требований к программному обеспечению. Но можно дать следующие общие рекомендации по выбору типов данных:

1. Для коротких символьных значений и символьных строк фиксированной длины следует выбирать тип CHAR. (NCHAR). Например, для поля "единица измерения" со значениями 'кг', 'шт.', 'уп.' (char (3)), для поля "пол" (char (1)) и т.п.
2. Для символьных строк переменной длины нужно выбирать тип VARCHAR (NVARCHAR) с указанием максимально возможной длины хранимого значения. Если при добавлении данных длина строки превысит указанное ограничение, система не сможет добавить данные и вернёт сообщение об ошибке.
3. Для числовых атрибутов, которые не участвуют в математических расчетах (номер телефона, номер паспорта) желательно использовать текстовое представление типа VARCHAR (NVARCHAR).
4. Для хранения дат нужно выбирать тип DATE или его варианты (DATETIME, например). Это позволит использовать арифметику дат и не заботиться о правильности вводимых данных: СУБД сама проверит допустимость даты.
5. Для семантически одинаковых полей разных таблиц нужно выбирать одинаковые типы данных. Например, фамилия сотрудника и фамилия клиента.

### Описание ограничений целостности

**Виды ограничений целостности**

Обеспечение целостности данных гарантирует отсутствие потерянных данных в базе данных.

Целостность данных подразделяется на следующие виды.

* Сущностная целостность
* Доменная целостность
* Ссылочная целостность
* Пользовательская целостность

**Сущностная целостность**

Сущностная целостность определяет строку как уникальную сущность в конкретной таблице. В таблице не должно быть одинаковых строк. Сущностная целостность может быть обеспечена с помощью первичного ключа таблицы, а если первичный ключ не используется, то с помощью полей, на которые наложено ограничение UNIQUE.

**Доменная целостность**

Доменная целостность — это принадлежность всех данных в домене (столбце) определенному типу и размеру данных и наложенным на столбец ограничениям с помощью оператора CHECK и опций FOREIGN KEY, DEFAULT, NOT NULL.

**Ссылочная целостность**

Ссылочная целостность сохраняет определенные связи между таблицами при добавлении или удалении строк. В SQL Server ссылочная целостность основана на связи первичных и внешних ключей и обеспечивается с помощью ограничений PRIMARY KEY и FOREIGN KEY. Ссылочная целостность гарантирует согласованность значений ключей во всех таблицах. Этот вид целостности требует отсутствия ссылок на несуществующие значения, а также обеспечивает согласованное изменение ссылок во всей базе данных при изменении значения ключа.

**Принудительное обеспечение целостности данных**

Планирование и создание таблиц требует указания допустимых значений для столбцов и определения способов принудительного обеспечения целостности данных в них. SQL Server предоставляет следующие механизмы для принудительного обеспечения целостности данных в столбце.

* [Ограничения PRIMARY KEY](https://technet.microsoft.com/ru-ru/library/ms191236(v=sql.105).aspx)
* [Ограничения FOREIGN KEY](https://technet.microsoft.com/ru-ru/library/ms175464(v=sql.105).aspx)
* [Ограничения UNIQUE](https://technet.microsoft.com/ru-ru/library/ms191166(v=sql.105).aspx)
* [Ограничения CHECK](https://technet.microsoft.com/ru-ru/library/ms188258(v=sql.105).aspx)
* [Определения DEFAULT](https://technet.microsoft.com/ru-ru/library/ms189909(v=sql.105).aspx)
* [Разрешение значений NULL](https://technet.microsoft.com/ru-ru/library/ms189265(v=sql.105).aspx)

Обеспечение целостности данных может осуществляться пользователем в прикладной программе или системой управления базами данных. Наиболее важными преимуществами предоставления ограничений целостности системой управления базами данных являются следующие:

* повышение надежности данных;
* сокращение времени на программирование;
* упрощение технического обслуживания.

Определение ограничений для обеспечения целостности посредством СУБД повышает надежность данных, поскольку устраняется возможность, что программист прикладного приложения может забыть реализовать их. Если ограничения целостности предоставляются прикладными программами, то все приложения, затрагиваемые этими ограничениями, должны содержать соответствующий код. Если код отсутствует хоть в одном приложении, то целостность данных будет поставлена под сомнение.

Если ограничения для обеспечения целостности не предоставляются системой управления базами данных, то их необходимо определить в каждой программе приложения, которая использует данные, включенные в это ограничение. В противоположность этому, если ограничения для обеспечения целостности предоставляются системой управления базами данных, то их требуется определить только один раз. Кроме этого, код для ограничений, предоставляемых приложениями, обычно более сложный, чем в случае таких же ограничений, предоставляемых СУБД.

Если ограничения для обеспечения целостности предоставляются СУБД, то в случае изменений ограничений, соответствующие изменения в коде необходимо реализовать только один раз — в системе управления базами данных. А если ограничения предоставляются приложениями, то модификацию для отражения изменений в ограничениях необходимо выполнить в каждом из этих приложений.

**Предложение DEFAULT**

Каждое поле записи должно содержать значение, даже если это значение равно NULL. Возможны случаи, когда необходимо поместить данные в таблицу, однако значение какого-либо поля неизвестно, или это значение не существует. Если столбец допускает значения NULL, то можно поместить в него значение NULL. Поскольку иногда помещать в столбцы значения NULL нежелательно, лучшим решением может оказаться определение для столбца значения по умолчанию (DEFAULT). Например, для числовых столбцов обычно в качестве значения по умолчанию указывается ноль.

Если столбец не допускает значений NULL и определение DEFAULT для него отсутствует, необходимо явно указать значение столбца, в противном случае компонент Database Engine возвратит сообщение об ошибке, указывающее, что значения NULL в столбце не допускаются.

Вставляемое в столбец значение определяется совместно определением DEFAULT и возможностью помещать в столбец значения NULL. Сводка результатов при различных комбинациях этих параметров приведена в следующей таблице.

|  |  |  |  |
| --- | --- | --- | --- |
| **Определение столбца** | **Запись отсутствует,**  **нет определения DEFAULT** | **Запись отсутствует,**  **есть определение DEFAULT** | **Ввод значения NULL** |
| Значения NULL допустимы | NULL | Значение по умолчанию | NULL |
| Значения NULL запрещены | Ошибка | Значение по умолчанию | Ошибка |

# Изменение и создание определений DEFAULT

Можно создать определение DEFAULT в определении таблицы при ее создании. Если таблица уже существует, можно добавить к ней определение DEFAULT. Каждый столбец таблицы может содержать одно определение DEFAULT.

Если определение DEFAULT уже существует, можно изменить его или удалить. Например, можно изменить значение, которое будет помещаться в столбец, если при вставке значение не указано. Для изменения определения DEFAULT необходимо сначала удалить существующее определение DEFAULT, а затем создать его повторно в новом определении.

Для столбцов, определения которых включают следующие элементы или свойства, создать определение DEFAULT невозможно:

* Тип данных timestamp.
* Разреженный столбец — поскольку разреженный столбец должен допускать значения NULL.
* Свойство IDENTITY или ROWGUIDCOL.
* Существующее определение DEFAULT или объект DEFAULT.

Когда определение DEFAULT добавляется к существующему в таблице столбцу, по умолчанию Database Engine использует это определение лишь для новых добавляемых в таблицу данных. Существующие данные, которые вносились во время действия предыдущего определения DEFAULT, остаются без изменений. Однако при добавлении нового столбца к существующей таблице, можно указать, чтобы вместо значения NULL Database Engine поместил значение по умолчанию (указанное в определении DEFAULT) в новый столбец во всех существующих в таблице строках.

После удаления определения DEFAULT Database Engine будет помещать в соответствующий столбец добавляемых строк значение NULL, а не значение по умолчанию, когда при вставке значение столбца не указано. Однако уже находящиеся в таблице данные не изменяются.

**Предложение UNIQUE**

Иногда несколько столбцов или группа столбцов таблицы имеет уникальные значения, что позволяет использовать их в качестве первичного ключа. Столбцы или группы столбцов, которые можно использовать в качестве первичного ключа, называются потенциальными ключами (candidate key). Каждый потенциальный ключ определяется, используя предложение UNIQUE в инструкции CREATE TABLE или ALTER TABLE. Синтаксис предложения UNIQUE следующий:

**[CONSTRAINT имя\_ограничения]**

**UNIQUE [CLUSTERED NONCLUSTERED ({имя\_столбца1},...)**

Опция CONSTRAINT в предложении UNIQUE присваивает явное имя потенциальному ключу. Опция CLUSTERED или NONCLUSTERED связана с тем обстоятельством, что компонент Database Engine создает индекс для каждого потенциального ключа таблицы. Этот индекс может быть кластеризованным, когда физический порядок строк определяется посредством индексированного порядка значений столбца. Если порядок строк не указывается, индекс является некластеризованным . По умолчанию применяется опция NONCLUSTERED. Параметр col\_name1 обозначает имя столбца, который создает потенциальный ключ. (Потенциальный ключ может иметь до 16 столбцов.)

**Предложение PRIMARY KEY**

Первичным ключом таблицы может являться столбец или группа столбцов, значения (набор значений для группы) которого разные в каждой строке. Каждый первичный ключ определяется, используя предложение PRIMARY KEY в инструкции CREATE TABLE или ALTER TABLE. Синтаксис предложения PRIMARY KEY следующий:

**[CONSTRAINT имя\_ограничения]**

**PRIMARY KEY ({имя\_столбца1},...)**

Все параметры предложения PRIMARY KEY имеют такие же значения, как и соответствующие одноименные параметры предложения UNIQUE. Но в отличие от столбца UNIQUE, столбец PRIMARY KEY не разрешает значений NULL и имеет значение по умолчанию CLUSTERED.

В примере 1 показано объявление первичного ключа для таблицы Depts базы данных Shop.

Пример 1. Определение первичного ключа

**USE Shop;**

**CREATE TABLE Depts (**

**deptId INT IDENTITY,**

**dept NVARCHAR(30) NOT NULL,**

**CONSTRAINT PK\_deptId PRIMARY KEY (deptId) );**

В результате выполнения кода в примере 1 создается таблица Depts, в которой определен первичный ключ. Первичный ключ таблицы определяется посредством декларативного ограничения для обеспечения целостности с именем PK\_deptId. Это ограничение для обеспечения целостности является ограничением уровня таблицы, поскольку оно указывается после определения всех столбцов таблицы Depts.

Пример 2 аналогичен примеру 1, за исключением того, что первичный ключ таблицы depts определяется как ограничение уровня столбца.

Пример 2. Определение ограничения уровня столбца

**USE Shop;**

**CREATE TABLE Depts (**

**deptId INT CONSTRAINT PK\_dept\_Id PRIMARY KEY IDENTITY,**

**dept NVARCHAR(30) NOT NULL);**

**Предложение FOREIGN KEY**

Внешним ключом (foreign key) называется столбец (или группа столбцов таблицы), содержащий значения, совпадающие со значениями первичного ключа в этой же или другой таблице. Внешний ключ определяется с помощью предложения FOREIGN KEY в комбинации с предложением REFERENCES. Синтаксис предложения FOREIGN KEY следующий:

**[CONSTRAINT имя\_ограничения]**

**[[FOREIGN KEY] ({имя\_столбца1},.)]**

**REFERENCES имя таблицы ({имя\_столбца2},.)**

**[ON DELETE (NO ACTION] CASCADE | SET NULL | SET DEFAULT}]**

**[ON UPDATE (NO ACTION | CASCADE | SET NULL | SET DEFAULT}]**

Предложение FOREIGN KEY явно определяет все столбцы, входящие во внешний ключ. В предложении REFERENCES указывается имя таблицы, содержащей столбцы, создающие соответствующий первичный ключ. Количество столбцов и их тип данных в предложении FOREIGN KEY должны совпадать с количеством соответствующих столбцов и их типом данных в предложении REFERENCES (и, конечно же, они должны совпадать с количеством столбцов и типами данных в первичном ключе таблицы, на которую они ссылаются).

Таблица, содержащая внешний ключ, называется ссылающейся (или дочерней) таблицей (referencing table), а таблица, содержащая соответствующий первичный ключ, называется ссылочной (referenced table) или родительской (parent table) таблицей.

В примере 3 показано объявление внешнего ключа для таблицы Sales базы данных Shop.

Пример 3. Объявление внешнего ключа

**USE Shop;**

**CREATE TABLE Sales (**

**recordId IDENTITY,**

**goodId INT NOT NULL,**

**quantity DECIMAL (6,2) NOT NULL,**

**price DECIMAL(6,2) NOT NULL,**

**saleId INT NOT NULL**

**CONSTRAINT PK\_recordId PRIMARY KEY (recordId),**

**CONSTRAINT FK\_goodId FROREIGN KEY (goodId)**

**REFERENCES Goods (goodId),**

**CONSTRAINT FK\_saleId FOREIGN KEY(saleId)**

**REFERENCES ledger (SaleId));**

Таблица sales в примере 3 задается с тремя декларативными ограничениями для обеспечения целостности: PK\_recordId, FK\_goodId, FK\_saleId. Все ограничения являются ограничениями уровня таблицы. Первое ограничение PK\_recordId определяет первичный ключ, а второе FK\_goodId и третье FK\_saleId — внешние ключи таблицы Sales (Продажи). Ограничение FK\_goodId определяет таблицу Goods (Товары) как ссылочную таблицу, а ее столбец goodId как соответствующий первичный ключ этой таблицы. Ограничение FK\_saleId определяет таблицу ledger (Журнал учета) как ссылочную таблицу, а ее столбец saleId как соответствующий первичный ключ этой таблицы.

Предложения ON DELETE и ON UPDATE не указаны, поэтому SQL SERVER устанавливает по умолчанию эти правила обеспечения ссылочной целостности в значения No Action. Это означает, что если при удалении или модификации данных возможно возникновение нарушения целостности данных, то это удаление или модификация не будет произведено и будет выдано соответствующее сообщение.

**Предложение CHECK**

Проверочное ограничение (check constraint) определяет условия, которым должны удовлетворять данные, вставляемые в столбец таблицы. Проверочные ограничения устанавливаются посредством предложения CHECK, определяемого в инструкции CREATE TABLE или ALTER TABLE. Синтаксис предложения CHECK следующий:

**[CONSTRAINT имя\_ограничения]**

**CHECK [NOT FOR REPLICATION] логическое\_выражение**

Параметр «логическое\_выражение» может ссылаться на любые столбцы в текущей таблице (или только на текущий столбец, если определен как ограничение уровня столбца), но не на другие таблицы.

В примере 4. показано применение предложения CHECK.

Пример 4. Применение предложения CHECK

**USE Shop;**

**CREATE TABLE Sales (**

**recordId IDENTITY,**

**goodId INT NOT NULL,**

**quantity DECIMAL (6,2) NOT NULL,**

**price DECIMAL(6,2) NOT NULL,**

**saleId INT NOT NULL**

**CONSTRAINT PK\_recordId PRIMARY KEY (recordId),**

**CONSTRAINT FK\_goodId FROREIGN KEY (goodId)**

**REFERENCES goods (goodId),**

**CONSTRAINT FK\_saleId FOREIGN KEY(saleId)**

**REFERENCES ledger (saleId),**

**CONSTRAINT CK\_quantity CHECK quantity>0,**

**CONSTRAINT CK\_price CHECK price>0);**

В таблице Sales (Продажи), создаваемой в примере 4 определены два столбца с проверочными ограничениями – столбец quantity (количество) и столбец price (цена). Значения, вставляемые в эти столбцы должны быть неотрицательными числами. При вставке отрицательного числа в эти столбцы или при попытке изменить значения в этих столбцах на отрицательные система управления базой данных выдаст сообщение об ошибке.

**Ссылочная целостность**

Ссылочная целостность (referential integrity) обеспечивает выполнение правил для вставок и обновлений таблиц, содержащих внешний ключ и соответствующее ограничение первичного ключа. Если для двух таблиц определены первичный и внешний ключи, то модифицирование значений в первичном ключе и соответствующем внешнем ключе будет не всегда возможным.

**Возможные проблемы со ссылочной целостностью**

Модификация значений внешнего или первичного ключа может создавать проблемы в четырех случаях.

**Случай 1**

Вставка новой строки в таблицу Goods (Товары) товара с номером группы 21.

Соответствующая инструкция Transact-SQL выглядит таким образом:

**USE Shop;**

**INSERT INTO goods**

**VALUES (‘яблоки’, 1, 21);**

При вставке новой строки в таблицу Goods (Товары) используется новый номер группы товаров, для которого нет совпадающего номера в родительской таблице Groups (Группы). Если для таблиц Groups (Группы) и Goods (Товары) определено правило обеспечения ссылочной целостность, то СУБД не допустит вставки новой строки с таким номером группы.

**Случай 2**

Изменение номера продажи 1 во всех строка таблицы Sales (Продажи) на номер 21.

Соответствующая инструкция Transact-SQL выглядит таким образом:

**USE Shop;**

**UPDATE Sales**

**SET saleId = 21 WHERE saleId = 1;**

В данном случае существующее значение внешнего ключа в ссылающейся таблице Sales (Продажи) заменяется новым значением, для которого нет совпадающего значения в родительской таблице Ledger (Журнал учета продаж). Если для обеих таблиц определено правило обеспечения ссылочной целостности, то система управления базой данных не допустит модификации строки с таким значением поля saleId в таблице Sales.

**Случай 3**

Замена значения номера группы (groupId) 2 в таблице Groups (Группы) на значение 22. Соответствующая инструкция Transact-SQL будет выглядеть таким образом:

**USE Shop;**

**UPDATE Groups**

**SET groupId = 22 WHERE groupId = 2;**

В данном случае предпринимается попытка заменить существующее значение поля groupId= 2

значением 22 только в родительской таблице Groups, не меняя соответствующие значения groupId в ссылающейся таблице Sales (Продажи). Система не разрешает выполнения этой операции. Правило обеспечения ссылочной целостности не допускает существования в ссылающейся таблице (таблице, для которой предложением FOREIGN KEY определен внешний ключ) таких значений, для которых в родительской таблице (таблице, для которой предложением PRIMARY KEY определен первичный ключ) не существует соответствующего значения.

В противном случае такие строки в ссылающейся таблице были бы "сиротами".

Если бы описанная выше модификация таблицы Groups была разрешена, тогда строки в таблице Sales со значением groupId равным 2 стали бы сиротами. Поэтому система и не разрешает выполнения такой модификации.

**Случай 4**

Удаление строки в таблице Groups со значением groupId =2. Этот случай похожий на случай 3. В случае выполнения этой операции, из таблицы Groups была бы удалена строка со значением groupId=2, для которого существуют совпадающие значения в ссылающейся (дочерней) таблице Sales. В результате таких действий все строки таблицы Sales, в которых значения groupId =2 стали бы “сиротами”. Поэтому СУБД не разрешает выполнения такого удаления.

**Опции ON DELETE и ON UPDATE**

При попытке обновления значения внешнего ключа, когда новое значение внешнего ключа не совпадает ни с одним из значений первичного ключа, СУБД откажется выполнять эти обновления и выведет сообщение об ошибке.

Но при попытке внести обновления в значения первичного ключа, вызывающие несогласованность в соответствующем внешнем ключе (случай 3 и 4 в предыдущем разделе), СУБД может реагировать достаточно гибко. Существует четыре опции, определяющих то, как система базы данных может реагировать.

* + - 1. **NO ACTION**. Модифицируются (обновляются или удаляются) только те значения в родительской таблице, для которых нет соответствующих значений во внешнем ключе дочерней (ссылающейся) таблицы, в противном случае удаление или модификация не происходит и выдается сообщение об ошибке.
      2. **CASCADE**. Разрешается модификация (обновление или удаление) любых значений в родительской таблице. При обновлении значения первичного ключа в родительской таблице (или при удалении всей строки, содержащей данное значение), в дочерней (ссылающейся) таблице обновляются (или удаляются) все строки с соответствующими значениями внешнего ключа.
      3. **SET NULL**. Разрешается модификация (обновление или удаление) любых значений в родительской таблице. Если обновление значения в родительской таблице вызывает несогласованность в дочерней таблице, то СУБД присваивает внешнему ключу всех соответствующих строк в дочерней таблице значение NULL, если такое значение разрешено в данном поле. Если значение NULL не разрешено, то СУБД не сможет создать связь с опцией SET NULL между родительской и дочерней таблицами. То же самое происходит и в случае удаления строки в родительской таблице, вызывающего несогласованность в дочерней таблице. Таким образом, все несогласованности данных пропускаются.
      4. **SET DEFAULT**. Аналогично опции SET NULL, но с одним исключением: всем внешним ключам, соответствующим модифицируемому первичному ключу, присваивается значение по умолчанию.

**2.1. Нормализация таблиц**

При проектировании таблиц возможно появление избыточного дублирования данных и аномалий.

Под избыточным дублированием данных будем понимать такое дублирование, от которого можно избавиться без потери функциональности.

Следует различать простое (неизбыточное) и избыточное дублирование данных. Наличие первого из них допускается в базах данных, а избыточное дублирование данных может приводить к проблемам при обработке данных.

Пример *неизбыточного дублирования* данных представляют данные, приведенные в таблице 14.4 со столбцами Сотрудник и Телефон. Для сотрудников, находящихся в одном помещении, номера телефонов совпадают. Номер телефона 4328 встречается несколько раз, хотя для каждого служащего номер телефона уникален. Поэтому ни один из номеров не является избыточным. Действительно, при удалении одного из номеров телефонов будет утеряна информация о том, по какому номеру можно дозвониться до одного из сотрудников.

Таблица 4. Таблица Сотрудник –Телефон с неизбыточным дублированием

|  |  |
| --- | --- |
| **Сотрудник** | **Телефон** |
| Иванов И.М. | 3721 |
| Петров М.И. | 4328 |
| Сидоров Н.Г. | 4328 |
| Егоров В.В. | 4328 |

Пример *избыточного дублирования (избыточности)* представляют данные приведенные в таблице 5 со столбцами Сотрудник – Телефон – Номер кабинета. Если в кабинете имеется один телефон, то все сотрудники, работающие в этой кабинете, имеют один и тот же номер телефона. В таблице этот номер указан трижды, следовательно, в этой таблице имеется избыточное дублирование данных.

Таблица 5. Таблица Сотрудник-Телефон – Номер кабинета с избыточным дублированием

|  |  |  |
| --- | --- | --- |
| **Сотрудник** | **Телефон** | **Номер кабинета** |
| Иванов И.М. | 3721 | 109 |
| Петров М.И | 4328 | 111 |
| Сидоров Н.Г. | 4328 | 111 |
| Егоров В.В. | 4328 | 111 |

Для устранения избыточности данных в этом случае можно создать дополнительную таблицу со столбцами Номер кабинета – Телефон (таблица 6), а в основной таблице удалить столбец Телефон, как показано в таблице 7.

Таблица 6. Таблица Номер кабинета - Телефон

|  |  |
| --- | --- |
| **Номер комнаты** | **Телефон** |
| 109 | 3721 |
| 111 | 4328 |

Таблица 7. Таблица Сотрудник – Номер кабинета

|  |  |
| --- | --- |
| **Сотрудник** | **Номер комнаты** |
| Иванов И.М. | 109 |
| Петров М.И. | 111 |
| Сидоров Н.Г. | 111 |
| Егоров В.В. | 111 |

Избыточное дублирование данных создает проблемы при обработке строк таблиц, называемые «аномалиями обновления таблицы».

***Аномалиями*** будем называть такую ситуацию в таблицах БД, которая приводит к противоречиям в БД либо существенно усложняет обработку данных.

Выделяют три основные вида аномалий:

* аномалии модификации (или редактирования);
* аномалии удаления;
* аномалии добавления.

***Аномалии модификации***проявляются в том, что изменение значения одного данного может повлечь за собой просмотр всей таблицы и соответствующее изменение некоторых других записей таблицы.

Так, например, изменение номера телефона в кабинете 111 (таблица 5), что представляет собой один единственный факт, потребует просмотра всей таблицы и изменения поля Номер комнаты в записях, относящихся к Петрову, Сидорову и Егорову.

***Аномалии удаления***состоят в том, что при удалении какого-либо данного из таблицы может пропасть и другая информация, которая не связана напрямую с удаляемым данным.

В той же таблице 5 удаление записи о сотруднике Иванове (например, по причине увольнения) приводит к исчезновению информации о номере телефона, установленного в 109-м кабинете.

***Аномалии добавления***возникают в случаях, когда информацию в таблицу нельзя поместить до тех пор, пока она неполная, либо вставка новой записи требует дополнительного просмотра таблицы.

Примером может служить операция добавления нового сотрудника все в ту же таблицу 5. Очевидно, будет противоестественным хранение сведений в этой таблице только о кабинете и номере телефона в ней, пока никто из сотрудников не помещен в нее. Более того, если в таблице 5 поле Сотрудник является ключевым, то хранение в ней неполных записей с отсутствующей фамилией сотрудника просто недопустимо из-за неопределенности значения ключевого поля.

Вторым примером возникновения аномалии добавления может быть ситуация включения в таблицу нового сотрудника. При добавлении таких записей для исключения противоречий желательно проверить номер телефона и соответствующий номер кабинета хотя бы с одним из сотрудников, сидящих с новым сотрудником в том же кабинете. Если же окажется, что у нескольких сотрудников, сидящих в одном кабинете, имеются разные телефоны, то вообще не ясно, что делать (то ли в комнате несколько телефонов, то ли какой-то из номеров ошибочный).

**Цель нормализации**: исключить избыточное дублирование данных, которое является причиной аномалий, возникающих при добавлении, редактировании и удалении строк таблицы.  
  
**Аномалии-модификации** проявляются в том, что изменение одних данных может повлечь просмотр всей таблицы и соответствующее изменение некоторых записей таблицы.  
  
**Аномалии-удаления** — при удалении какой либо строки из таблицы может пропасть информация, которая не связана на прямую с удаляемой записью.  
  
**Аномалии-добавления** возникают, когда информацию в таблицу нельзя поместить, пока она не полная, либо вставка записи требует дополнительного просмотра таблицы.

**Функциональные зависимости.**

Реляционная база данных содержит как структурную, так и семантическую информацию. Структура базы данных определяется числом и видом включенных в нее таблиц и связями, существующими между ними. Семантическая часть описывает множество функциональных зависимостей, существующих между атрибутами этих таблиц.

**Определение:**

Если даны два атрибута *X* и *Y* некоторой таблицы, то говорят, что *Y* функционально зависит от *X*, если в любой записи этой таблицы значению *X* соответствует ровно одно значение *Y*.

Функциональная зависимость обозначается *X -> Y*. Отметим, что *X* и *Y* могут представлять собой не только единичные атрибуты, но и группы, составленные из нескольких атрибутов одной таблицы. Можно сказать, что функциональные зависимости представляют собой связи, существующие внутри таблицы.

Некоторые функциональные зависимости могут быть нежелательны.

**Определение:**

Избыточная функциональная зависимость - зависимость, заключающая в себе такую информацию, которая может быть получена на основе других зависимостей, имеющихся в базе данных.

Корректной считается такая схема базы данных, в которой отсутствуют избыточные функциональные зависимости. В противном случае приходится прибегать к процедуре декомпозиции (разложения) имеющегося множества таблиц. При этом порождаемое множество содержит большее число таблиц, которые являются проекциями таблиц исходного множества.  *Обратимый* пошаговый процесс замены данной совокупности таблиц другой схемой с устранением избыточных функциональных зависимостей называется **нормализацией**.

Условие обратимости требует, чтобы декомпозиция сохраняла эквивалентность схем при замене одной схемы на другую, т.е. в результирующих таблицах:

* не должны появляться ранее отсутствовавшие записи;
* на таблицах новой схемы должно выполняться исходное множество функциональных зависимостей.

**2.1.1   Первая нормальная форма (1NF).**

Для обсуждения первой нормальной формы необходимо дать два определения:

**Простой атрибут** - атрибут, значения которого атомарны (неделимы).

**Сложный атрибут** – атрибут, полученный соединением нескольких атомарных атрибутов, которые могут быть определены на одном или разных доменах.

**Определение первой нормальной формы:**

Таблица находится в 1NF если значения всех ее атрибутов атомарны.

В базе данных отдела кадров предприятия необходимо хранить сведения о служащих, которые можно попытаться представить в отношении

РАБОТНИК(Табельный номер, Фамилия – имя- отчество, Дата рождения).

В этой таблице атрибут *"Табельный номер"*  является атомарным, а атрибуты ФИО и Дата рождения  являются сложными, так как каждый из них может быть разделен на три атрибута:

ФИО – Фамилия, Имя, Отчество

Дата рождения – Год, месяц, день

Для приведения исходной таблицы РАБОТНИК к первой нормальной форме необходимо представить таблицу со следующими атрибутами:

РАБОТНИК(Табельный номер, Фамилия, Имя, Отчество, Год рождения, Месяц рождения, День рождения)

Табельный номер является уникальным атрибутом и он может быть выбран первичным ключом или наоборот в таблицу может быть введен идентификатор записи (ID), значения которого задаются счетчиком и он может быть использован в качестве табельного номера.

**2.1.2. Вторая нормальная форма (2NF).**

Иногда первичный ключ таблицы делается составным – состоящим из нескольких атрибутов.

При этом вводится понятие **полной функциональной зависимости**.

**Определение:** неключевой атрибут функционально полно зависит от составного ключа если он функционально зависит от всего ключа в целом, но не находится в функциональной зависимости от какого-либо из входящих в него атрибутов.

**Определение второй нормальной формы:**

Таблица находится во 2НФ, если она находится в 1НФ и каждый неключевой атрибут функционально полно зависит от ключа.

Пример 1 :

R = { ’Название группы’, ’Название СД-диска’, ’Название песни’, 'Автор слов’, ’Композитор’ }

Одна и та же песня может входить в несколько дисков, также теоретически возможны одноименные альбомы с одноименными песнями у разных групп. Поэтому первичным ключом может быть  **{ ’Название группы’, ’Название СД-диска’, ’Название песни’ }**. При этом атрибуты **’Автор слов’** и **’Композитор’** зависят от множества атрибутов **{ ’Название группы’, ’Название песни’ },** то есть от части первичного ключа, а не от всего ключа . Это и есть нарушение 2NF.

Пример 2: На складах хранятся детали. Имеется таблица вида  
R={Деталь, Склад, Количество, Адрес склада}

Ключом в этой таблице может быть пара **{ ’Деталь’, ’Склад’ }**, но **’Адрес\_склада’** функционально зависит только от атрибута **’Склад’**, то есть от подмножества атрибутов ключа. Поэтому требования 2NF несоблюдены.

Чем это плохо?

Во-первых, адрес склада будет дублироваться для всех деталей на складе (избыточность) и если адрес изменится, нужно будет изменить все эти записи, чтобы сохранить целостность (могут возникнуть аномалии удаления).

Во-вторых, если на складе еще нет деталей, то у нас нет возможности хранить адрес склада, так как схема такую ситуацию не предусматривает. Поэтому добавление нового склада невозможно (аномалия вставки), а вывоз деталей со склада означает, что мы потеряем информацию о его адресе (аномалия удаления).

* + 1. **Tретья нормальная форма 3NF .**

***Определение.*** Базовое отношение находится в **третьей нормальной форме**  тогда и только тогда, когда оно находится во второй нормальной форме и каждый неключевой атрибут полностью функционально зависит только от ключей.

Третья нормальная форма предполагает, что каждый столбец, не являющийся ключом, должен зависеть **только** от столбца, который является ключом, то есть должна отсутствовать **транзитивная функциональная зависимость** (transitive functional dependency)

Транзитивная функциональная зависимость выражается следующим образом: А → В и В → С. То есть атрибут С транзитивно зависит от атрибута А, если атрибут С зависит от атрибута В, а атрибут В зависит от атрибута А (при условии, что атрибут А функционально не зависит ни от атрибута В, ни от атрибута С).

Если столбец зависит не только от первичного ключа, то данный столбец находится не в той таблице, в которой он должен находиться, либо же является производным от других столбцов.

Для нормализации из исходной таблицы те атрибуты, которые находятся в транзитивной зависимости от ключа, выносятся в отдельную таблицу с копией того атрибута, от которого они непосредственно зависят.

При применении третьей нормальной формы таблица должна находиться во второй нормальной форме. 3NF позволяет значительно снизить избыточность данных.

Пример

Сотрудники (**№ табельный**, Фамилия, Имя, Отчество, Код должности, Оклад);

Primary key (№ табельный);

Кроме того, над данным отношением «Сотрудники» задана следующая система функциональных зависимостей:

{Код должности} ? {Оклад};

Действительно, как правило, от должности, а следовательно, от ее кода в соответствующей базе данных напрямую зависит размер оклада, т. е. размер заработной платы.

Именно поэтому это отношение «Сотрудники» и не находится в третьей нормальной форме, ведь получается, что неключевой атрибут «Оклад» полностью функционально зависит от атрибута «Код должности», хотя этот атрибут и не является ключевым.

* + 1. **Нормальная форма Бойса-Кодда BCNF.**

Эта нормальная форма вводит дополнительное ограничение по сравнению с 3NF.   
**Определение нормальной формы Бойса-Кодда:**

Базовое отношение находится в **нормальной форме Бойса** – **Кодда** тогда и только тогда, когда она находится в третьей нормальной форме, и при этом не только любой неключевой атрибут полностью функционально зависит от любого ключа, но и любой ключевой атрибут должен полностью функционально зависеть от любого ключа.

Ситуация, когда отношение будет находится в 3NF, но не в BCNF, возникает при условии, что отношение имеет два (или более) возможных ключа, которые являются составными и имеют общий атрибут. Заметим, что на практике такая ситуация встречается достаточно редко, для всех прочих отношений 3NF и BCNF эквивалентны.

**Определение четвертой нормальной формы:**

Отношение находится в 4NF если оно находится в BCNF и в нем отстутсвуют многозначные зависимости, не являющиеся функциональными зависимостями.

**2.1.5  Зависимости по соединению и пятая нормальная форма (5NF).**

До сих пор мы предполагали, что единственной операцией, необходимой для устранения избыточности в отношении, была декомпозиция его на две проекции. Однако, существуют отношения, для которых нельзя выполнить декомпозицию без потерь на две проекции, но которые можно подвергнуть декомпозиции без потерь на три (или более) проекций. Этот факт получил название **зависимости по соединению**, а такие отношения называют **3-декомпозируемые отношения** (ясно, что любое отношение можно назвать "n-декомпозируемым", где n >= 2).

Зависимость по соединению является обощением многозначной зависимости. Отношения, в которых имеются зависимости по соединению, не являющиеся одновременно ни многозначными, ни функциональными, также характеризуются аномалиями обновления. Поэтому, вводится понятие пятой нормальной формы.

**Определение пятой нормальной формы:**

Отношение находится в 5НФ тогда и только тогда, когда любая зависимость по соединению в нем определяется только его возможными ключами.

Другими словами, каждая проекция такого отношения содержит не менее одного возможного ключа и не менее одного неключевого атрибута.

*До сих пор мы предполагали, что единственной операцией, необходимой для устранения избыточности в отношении, была декомпозиция его на две проекции. Однако, существуют отношения, для которых нельзя выполнить декомпозицию без потерь на две проекции, но которые можно подвергнуть декомпозиции без потерь на три (или более) проекций. Этот факт получил название****зависимости по соединению****, а такие отношения называют****3-декомпозируемые отношения****(ясно, что любое отношение можно назвать "n-декомпозируемым", где n >= 2).*

*Зависимость по соединению является обобщением многозначной зависимости. Отношения, в которых имеются зависимости по соединению, не являющиеся одновременно ни многозначными, ни функциональными, также характеризуются аномалиями обновления. Поэтому, вводится понятие пятой нормальной формы.*

***Определение пятой нормальной формы:***

*Отношение находится в 5НФ тогда и только тогда, когда любая зависимость по соединению в нем определяется только его возможными ключами.*

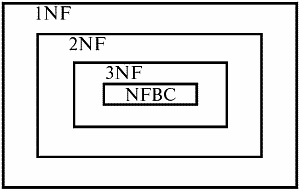
*Другими словами, каждая проекция такого отношения содержит не менее одного возможного ключа и не менее одного неключевого атрибута.*

**Вложенность нормальных форм**

Что означает вложенность нормальных форм друг в друга?

**Вложенность нормальных форм** – это отношение понятий ослабленной и усиленной формы по отношению друг к другу.

Вложенность нормальных форм полностью следует из их соответствующих определений. Диаграмма, иллюстрирующая отношение вложенности известных нам нормальных форм представлена ниже:



Первая нормальная форма является ослабленной по отношению ко второй нормальной форме (да и по отношению ко всем остальным нормальным формам тоже). Действительно, вспоминая определения всех рассмотренных нормальных форм, можно заметить, что требования каждой нормальной формы включали в себя требование принадлежности именно к первой нормальной форме (ведь она входила в каждое последующее определение).

Вторая нормальная форма является усиленной по отношению к первой нормальной форме, но ослабленной по отношению к третьей нормальной форме и нормальной форме Бойса – Кодда. На самом деле принадлежность второй нормальной форме включается в определение третьей, а сама вторая форма, в свою очередь, включает в себя первую нормальную форму.

Нормальная форма Бойса – Кодда является усиленной не только по отношению к третьей нормальной форме, но также и по отношению ко всем остальным, предшествующим ей.

А третья нормальная форма, в свою очередь, является ослабленной только по отношению к нормальной форме Бойса – Кодда.

### 

### Физическое проектирование БД

Этап физического проектирования заключается в увязке логической структуры БД и физической среды хранения с целью наиболее эффективного размещения данных, т.е. отображении логической структуры БД в структуру хранения. Решается вопрос размещения хранимых данных в пространстве памяти, выбора эффективных методов доступа к различным компонентам "физической" БД.

При физическом проектировании определяются следующие параметры базы данных

Имя базы данных

Имена и пути файла базы данных и файла журнала транзакций

Начальные размеры и шаг приращения файлов базы данных и журнала транзакций

При необходимости определяются количество файловых групп и места их хранения.

**Задание на лабораторную работу**

Разработать концептуальную, логическую и физическую модели базы данных для формирования информационной системы в предметной области в соответствии с вариантом.

В базе данных предусмотреть:

1. Не менее 5 таблиц.
2. Представить таблицы в 3НФ.
3. Предусмотреть средства обеспечения целостности базы данных.
4. Не менее 3 хранимых процедур.
5. Не менее 3 представлений.
6. Не менее 3 функций пользователя.
7. Не менее 3 триггеров.

**Варианты предметных областей.**

1. Кафедра – Информационная система учета оценок студентов по дисциплинам.
2. Автобаза – Информационная система учета работы транспорта и водителей.
3. Касса театра – Информационная система бронирования и продажи билетов.
4. Турфирма – Информационная система продажи туров.
5. Гостиница –Информационная система учета заселения и освобождения номеров.
6. Поликлиника – Информационная система регистратуры
7. Больница – Информационная система приемного отделения.
8. Отдел кадров – Информационная система начальника отдела.
9. Аэропорт – Информационная система прибытия и убытия самолетов
10. Авиакомпания – Информационная система продажи билетов.
11. ГАИ – Информационная система регистрации автомобилей.
12. Интернет-магазин – Информационная система учета продаж.
13. Интернет-магазин – Информационная система учета закупок
14. Ресторан – Информационная система учета продаж.
15. Ресторан – Информационная система учета закупок продуктов.
16. Банк – Информационная система учета вкладов
17. Банк – Информационная система учета кредитов
18. Отделение милиции – Информационная система учета происшествий.
19. ЗАГС – Информационная система учета оказания услуг.
20. Библиотека – Информационная система учета книжного фонда.
21. Рекламное агентство – Информационное агентство учета заказов и услуг
22. Компьютерная фирма – Информационная система учета закупки комплектующих.
23. Компьютерная фирма – Информационная система учета продажи товаров и услуг.
24. Парикмахерская – Информационная система учета закупок товаров и продаж услуг
25. Кондитерская – Информационная система учета закупок и продаж.
26. СТО – Информационная система учета закупок и продаж.
27. Автосалон – Информационная система учета доходов и расходов.
28. Общежитие – Информационная система расселения студентов.
29. Автозаправка – Информационная система учета продаж и поступления топлива.
30. Музей– Информационная система учета поступлений и отгрузки экспонатов.

**Содержание отчета**

1. Тема и цель работы
2. Задание на лабораторную работу
3. Концептуальная модель базы данных.
4. Логическая модель базы данных.
5. Физическая модель базы данных
6. Перечень представлений, хранимых процедур, функций пользователя и

триггеров с указанием их назначения.

1. Выводы

**Контрольные вопросы**

1. Что такое концептуальная модель базы данных?
2. Что такое логическая модель базы данных?
3. Что такое физическая модель базы данных?
4. Какие ограничения используются в базах данных?
5. Что такое первичный ключ и для чего он используется?
6. Что такое внешний ключ и для чего он используется?
7. Какие требования предъявляются к таблице в 1НФ (2НФ, 3НФ)?
8. Что такое сущность в предметной области?
9. Какие возможны виды связей между сущностями?
10. Какие существуют ограничения целостности базы данных?
11. Какие существуют правила обеспечения ссылочной целостности и что они означают?