

# Проектиране на БД

## **Теоретични основи**

Целта на настоящото упражнение е да се запознаят студентите с процеса на проектиране на БД. Основните процеси, които съпътстват проектирането, са:

1. Дефиниране на параметрите на системата – тук се определя какво трябва да се реализира, защо и как.
2. Дефиниране на работните процеси – основната задача на БД е поддържането на един или повече работни процеси. Тук се определя начина, по който ще се ползват данните, което от своя страна е свързано с разбиране на семантиката на модела данни.
3. Изграждане на концептуален модел на БД – концептуалният модел дефинира използването на данни за цялата система.
4. Определяне на схемата на БД – тя служи за преобразуване на концептуалния модел на БД във физически. Включва описание на таблиците, които ще се реализират в системата и физическата архитектура на данните.

## **Дефиниране на параметрите на системата**

Този процес включва три стъпки:

- Определяне на целите на проекта като цяло;
- Получаване на критериите, на които трябва да отговаря системата;
- Дефиниране на обхвата на системата.

## **Определяне на целите**

Много често дефиницията на целите и обхвата на системата е сложен процес, комбиниращ анализ, дизайн и малко дипломатия.

Най-важният фактор е определяне на целта на проекта. Той служи като основа за получаване обхвата и критериите, на които трябва да отговаря системата. Целта се определя от множеството причини, които налагат създаване на система. За да се определи целта, трябва да се отговори на въпроси от вида:

- Защо се налага изграждане на нова система?
  - За увеличаване на бързодействието
  - За намаляване на разходите
  - За подобряване на обслужването на клиентите

Целите са осезаеми и неосезаеми в зависимост дали имат количествен измерител. Добре е да се търсят осезаемите. Трябва да се обърне внимание когато целта касае подобрене или намаление. В тези случаи задължително трябва да има еталон за сравнение или граница, в която трябва да се вмести. Например, повишение на ефективността - с колко? Добре е в отговора да се съдържа измервателен елемент, например: „Да се намали времето за изработка от 10 мин. На 5 мин.“, а не „Да се намали времето за изработка двойно“.

Определянето на целта става с интервюта с клиента. Необходимо е да се разберат основните термини, с които борови той и как те се отнасят към целта на системата. Не считайте, че целта се определя само веднъж и остава непроменена до

края на проекта. В последващите фази на проекта може да претърпи изменение. Всеки по-сложен проект налага такава промяна.

## **Определяне критериите на дизайн на проекта**

След като сме приключили с определяне на целите, се пристъпва към разработката на критериите на проекта. Обикновено този етап се изпълнява заедно с първия, защото е пряко свързан с него. Ако целите определят накъде се отива, критериите определят кога да се спре. Всички критерии се привързват към една или повече цели като ги поддържат. Не трябва да се бъркат двете понятия или да се сливат.

Критериите имат една от трите форми:

- Директно измерими изисквания от вида: „Отпечатай отчета за по малко от 2 часа”.
- Критерии на средата като „Използвай текущата мрежа”.
- Стратегии на проекта от вида „Да е налична контекстно-чувствителна помощ”.

Всички критерии, които са получени трябва да се категоризират с цел да се провери дали са определени добре целите на системата. Повечето критерии попадат в първата или втората категории. Ако има само критерии от тип 3, то е необходимо отново да се премине през стъпка 1 (определяне на целите). Препоръчва се критериите от тип 3 да се реализират със съществуващи софтуерни средства (напр. MTS за организация на стабилна работа с транзакции).

- **Директно измерими изисквания**

Те пряко зависят от измеримите цели. Например, ако целта е да се намали времето с 50% и текущият процес има време за изпълнение 10 сек., то подходящо изискване би звучало като „Да се осъществи изпълнение на процеса за време по-малко или равно на 5 сек.”. При определяне то на директно измеримите изисквания не трябва да се променят изискванията. Например, ако за специален процес е дадено време за изпълнение 1 мин., то заявката, с която се изпълнява трябва да има време за изпълнение по-малко от 10 сек.

- **Критерии на средата**

Те включват OS и всички други софтуерни средства, с които трябва да си взаимодейства системата. Ограничение се налага и от съществуващият хардуер. Друг съществуващ критерий, специално за БД е обема на данните, които се обработват. При това трябва да се обърне внимание на абсолютния обем на данните и размера на увеличението им. Например, система за библиотека може да има 1 милион записа и всеки ден да се добавят по няколко нови. Друг пример е система за продажба, при която всеки ден може да се добавят хиляди записи, но след приключване на продажбата, те да се архивират и абсолютния обем данни да не е по-голям от няколко хиляди. Двата примера имат различни критерии.

При определяне на обема на данните се използва поне 10% резерв, като за по-малки системи този резерв нараства до 20%-25%. Резервът се определя спрямо максималните изисквания на клиента. Това се налага поради факта, че добре проектирана система, ако обслужва 100 000 записа, ще се справи и с 10 000, докато обратното не е вярно.

Друг важен критерий на средата е броя на потребителите, които се поддържат от системата. Почти всяка система има повече от една категория потребители, които трябва да обслужва. Често групите са:

- Оператори, въвеждащи данни.
- Оператори, изискващи справки.
- Оператори, актуализиращи данни.
- Администратори на системата и др.

Трябва да се прави разлика между потребители, които са включени в системата и тези, които действително я ползват. Jet DB Engine има лимит от 255 потребители, свързани по едно и също време. Това не означава, че 255 потребители ще могат да актуализират БД едновременно, а само че могат да я отворят.

- **Общи дизайн стратегии**

Някои цели не могат да получат количествен показател, например, „Да се подобри точността при въвеждане на данни”. Това е пример, в който разходите по измерване са големи и ще превишат бюджета на проекта. Какво се случва с тези цели? Те не се игнорират, но и не се търси начин за количествена оценка. За тях се определят общи дизайн стратегии. В горният пример, такава стратегия би звучала „Да се подобри точността при въвеждане на данни чрез допускане потребителите да избират данни от списък, където е възможно” или „Да се намалят инцидентите на кредитните изключителни ситуации, чрез реализиране на подходящи проверки на кредита преди запис на поръчката”.

При определяне на стратегиите следва да се избягват изречения от вида: „Системата трябва да бъде user friendly.”.

## **Определяне обхвата на системата**

Обхватът на системата трябва да се определи след дефиниция на целите. Съществуват две ситуации, които нарушават това правило:

1. Когато функцията на системата е ясна за потребителите и времето за разработката ѝ ще доведе до превишаване на общото време. В този случай се разширяват целите на системата.
2. Когато клиентът настоява за включване на нова функционалност.

- **Cost-Benefit Analysis (CBA)**

Той се прави за по-големи системи. Определя реда на създаване на системата, в зависимост от времето за разработка. Най-напред се разработват компонентите с най-голяма ползва за клиента. Това е времеемък процес.

Съществуват много начини за направа на този анализ. За всички функции се определя частното полза/време за разработка. Колкото е по-голяма тази стойност, толкова е по-важен компонента. Вместо време за разработка може да се ползва цена. За всеки компонент се използва един и същ показател (време или цена). Посложен в въпросът с измерване на ползата. Един от начините е да се оцени процентът на повишение на ефективността, или процентът на намаление на грешката, стига да може да се определи върху каква база ще се изчисли той. Когато това не е възможно се използват следните понятия: „спестени средства”, „печалба” и „неосезаеми ползи”. Изчисляват се и трите показателя за всяка функционалност, след което посредством нормализация се определя единствен показател. Ако и трите показания са еднакво важни, може да се изчисли средно аритметично.

Когато трите фактора са с различно тегло. Се въвежда теглови фактор на

важност, който се изчислява по 
$$\frac{\sum_{i=1}^3 (p_i * k_i)}{3}$$
, където  $p_i$  е  $i$ -тия показател, а  $k_i$  –  $i$ -тия теглови коефициент. Предпочита се вторият метод като по-точен.

СВА е удобно средство за определяне на ползите от системата. Дава възможност за сравнение между показателите. Резултатите от него трябва да се оценяват заедно с други фактори, като зависимости в системата.

#### **Зад. 1. Попълнене таблицата.**

Функц.	Спестени средства		Печалба		Неосезаеми ползи		Общо	Общо претегл.	Средно	Средно претегл.
	Коеф	Ед	Коеф	Ед	Коеф	Ед				
F1	1	6	4	3	2	2				
F2	1	2	4	6	2	3				

### **Дефиниране на работните процеси**

Работните процеси са дейностите, които трябва да поддържа системата. Работният процес е множество от една или повече задачи, които заедно представят дейност, полезна за клиента. Примери за работни процеси: „обработка на заявка за покупка“, „търсене на телефонен номер на клиент“. Примери за задачи: „запиши заявка за продажба“, „провери кредитната карта на клиент“. Понякога е трудно да се различи задача от дейност. В различните системи по различен начин се дефинира дейност и задача. Определяне на последователността зависи от семантиката на проблема, който се решава.

Някои системи не се нуждаят от анализ на работните процеси, например: ad hoc генератор на отчети. Той не поддържа специфични процеси. В такъв случай е по подходящо да се изградят потребителски сценарии.

### **Определяне на текущите работни процеси**

Първата стъпка в анализиране на бизнес-процесите е определяне на обхвата на системата. Редът, в който се определят процесите не е важен. Използват се следните стъпки:

- **Разговори с потребителите**

След като се определят работните процеси, произлизащи от обхвата на системата, трябва да се съществуват решения. За тази цел се провеждат разговори с потребителите. Преглеждат се всички документи, които се ползват, всички изходни справки, отчети, съществуващи системи, файлове на Excel, документи на Word, екранни форми и т.н. На всеки от тях се прави копие и върху него се водят бележки като се интервюират потребителите.

Добре е да се провеждат интервюта с потребители, участващи в работния процес и отделно с шефовете, за да се добие представа за спецификата на задачите, които системата трябва да извършва като цяло.

При интервюиране трябва да се обхождат всички изключителни ситуации на процеса. Трябва да се изясни и как се получават данните в работния процес: в коя

форма участват, какво става, ако са грешни или непълни, как се обработват. (За повече информация погледнете концептуалният модел на БД.)

Много работни процеси се състоят от задачи, изпълнени от различни хора. В такъв случай трябва да се проведе интервю с всеки един от тях. Интервюта трябва да се проведат и с хора, които привидно са извън обхвата на системата, но взаимодействат с нея. За всеки отчет трябва да се провери дали хората, които го ползват имат изисквания.

- **Идентификация на задачите**

След като е събрана информация от потребителите, следва организирането ѝ в задачи. Ключов момент е определяне на бизнес правилата, които се прилагат в този процес.

Бизнес правило е ограничение, породено от проблемната област, а не от типа данни. Например, „няма заявки с дата 31 април” не е бизнес правило. Пример за бизнес правило е „няма заявки с дата по-малка от текущата”. Терминът „бизнес правило” се използва, независимо дали става дума за бизнес организация или не. Най-важните бизнес правила се отнасят до данните. Пример за такива са: „пощенският код не може да е празен”, „датата на фактурата трябва да е по-голяма или равна на датата на заявката”. За да бъдат валидни бизнес правилата се налага групиране на дейностите. Пример: „Заявка за продажба” включва следните задачи:

1. Проверка на заявката дали всички данни са попълнение.
2. Получаване на информация за клиента, ако съществува.
3. Запис на информация за превоза на стоката.
4. Въвеждане на детайлите на поръчката.
5. Присвояване на номер на нов клиент.
6. Проверка дали стоките са налични.
7. Проверка дали кредитния лимит на клиента не е превишен.
8. Избор на поръчка.
9. Пакетиране на стоките.
10. Подготовка на документи за превоз.

Нека анализираме всяка точка. Т.1 има ясно начало и край. Тя започва, когато се получи нова поръчка и свършва, когато целия документ е проверен. Можем да допуснем, че бизнес правилата са валидни в началото на процеса или документа се отхвърля. Т.1 отговаря на критериите за задача и я приемаме като такава.

Т.2 съдържа само едно бизнес правило: достъп до данните за клиент. Ако друга задача в момента работи с тях, те ще са недостъпни. При това не са изпълнени изискванията за задача. Действието започва след проверката на поръчката, но не е ясен неговия край (зависи дали ще са свободни данните за клиента). Следователно това е част от задача, нейна стъпка.

Данните за клиента се ползват в т.3 и т.5, което е предпоставка да бъдат в една задача с т.2. Т.4 е част от процеса „запис на поръчка”. Тези точки не могат да се изпълнят, ако не е изпълнена т.7 и следователно т.7 спада към „запис на поръчка”.

Т.6 представя ясна задача, която започва след „запис на поръчка” и завършва с потвърждение за достатъчна наличност от стоки. Ако в бизнес правилото пише, че поръчка не може да се приеме, ако няма налични стоки, то т.6 преминава към задача „запис на поръчка”.

Анализът на т.8, т.9 и т.10 зависи от обхвата на системата. Ако доставката на стоки е част от системата, то тези три точки могат да се обединят в задача „доставка на стоки”.

**Задача:** По така представените указания преработете списъка от задачи.

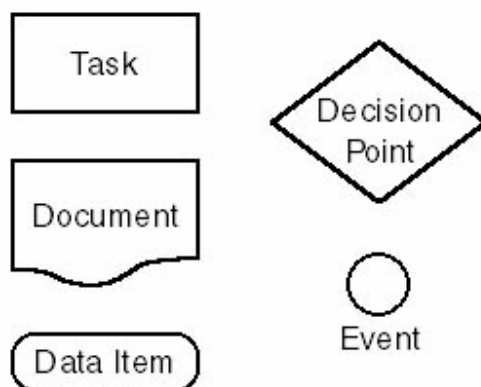
### Анализ на работните процеси

Трябва да се провери дали е точен реда на изпълнение на задачите в работния процес. При това се определя зависимостта между отделните задачи. Важно е да се прегледа зависимостта между данните. Някои задачи са отговорни за създаване на информация като номер на клиент и се използват от други задачи. Заедно с това трябва да се отделят задачите, които не е необходимо да се изпълняват.

### Документиране на работните процеси

След анализа се налага нова среща с клиентите, за да се провери доколко точно са съставени работните процеси. Ако проектът е сложен, то броят на срещите е много по-голям.

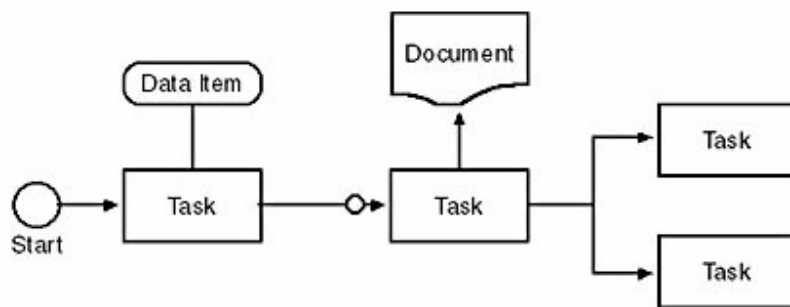
За документиране се ползва ER (Entity-Relationship) диаграма, Data flow диаграма и Process quality диаграма. Основните елементи за описание на работния процес са:



Когато задачата има малък брой стъпки може да се изобрази с правоъгълник, а при по-голям брой е добре да се направи отделна диаграма за всяка задача. Когато задачата се изпълнява от външен изпълнител се означава с удебелен правоъгълник.

Data Item (даннови елементи) представя атрибут (име на клиент, напр.) или обект (entity). Когато обектът е създаден от задача се изобразява удебелен или със сянка.

Елементите на работния процес се свързват помежду си със стрелки. При незадължителни задачи се ползва . Ето примерна workflow диаграма:



## Сценарий на потребителите

Състои се от два компонента:

- User profile – определя различни типове потребители, които ползват системата.
- Usage scenarios – за всеки профил на потребител се представя описание на това как потребителя очаква да взаимодейства със системата.

Сценариите дублират анализа на работните процеси, но от различен ъгъл. Понякога е трудно да се създаде сценарий, субективно е. Тук основно се включват очакванията на потребителя, неговата цел, а не транзакциите и стъпките в обработката на информация.

## Концептуален модел на данните

Това е следващата стъпка. Концептуалният модел съдържа описание на всеки елемент, атрибут и връзките между тях. Това не е схема на БД, която описва таблиците на физическо ниво. Преди създаване на схемата, трябва да е ясно как ще се реализира системата.

## Определяне на данните обекти

Първата стъпка е преглед на всички документи и екранни форми и създаване на работни процеси. Започва се с разглеждане на работния процес, който е основен. За всяка стъпка и задача се определят входно-изходните документи и екранни форми. Разглеждат се и от тях се генерира списък от атрибути, като се отбелязва кои се повтарят. Добавят се и данните елементи, открити и използвани от работния процес.

**Задача:** За показания по-долу екран определете атрибутите.



**NORTHWIND**  
TRADERS

## SALES ORDER

One Portals Way, Twin Points WA 98156  
Phone: 1-206-555-1417 Fax: 1-206-555-5938

Date: 26-May-99

**Ship To:** Alfreds Futterkiste  
Obere Str. 57  
Berlin 12209  
Germany

**Bill To:** Alfreds Futterkiste  
Obere Str. 57  
Berlin 12209  
Germany

Order ID:	Customer ID:	Salesperson:	Order Date:	Required Date:	Shipped Date:	Ship Via:
10643	ALFKI	Michael Suyama	25-Sep-95	23-Oct-95	03-Oct-95	Speedy Express

Product ID:	Product Name:	Quantity:	Unit Price:	Discount:	Extended Price:
28	Rössle Sauerkraut	15	\$45.60	25%	\$513.00
46	Spegesild	2	\$12.00	25%	\$18.00

**Subtotal:** \$531.00


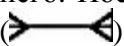
**Freight:** \$29.46

**Total:** \$560.46

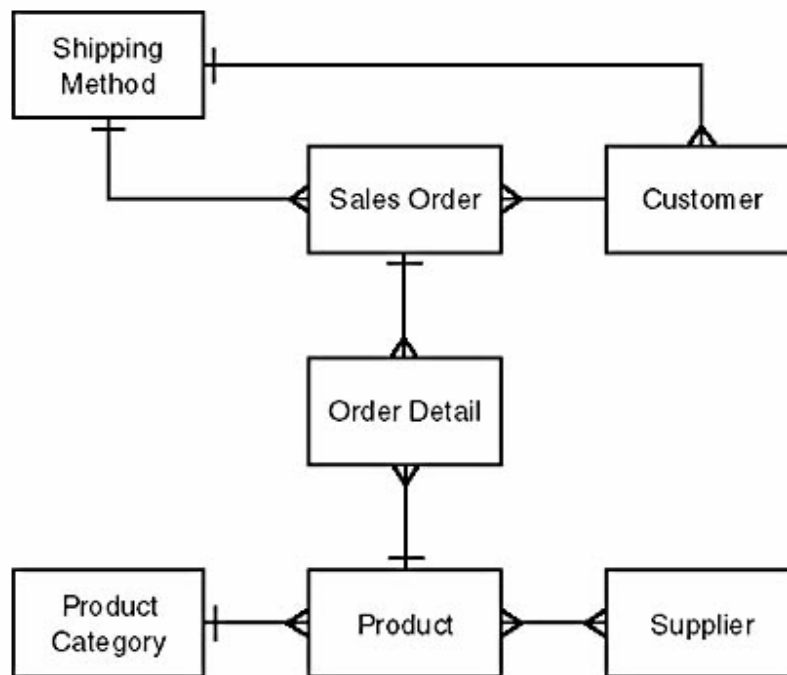
Следва определяне на атрибутите, обектите и връзките между тях. За всеки елемент от списъка се определя кога елемента е обект или задава факт за обекта. Обектите са entity, а фактите са атрибутите им.

**Задача: Определете фактите, обектите и връзките между тях.**


### Дефиниране на връзки

След определяне на атрибутите и обектите следва получаване на връзките между тях. За тази цел се построява ER диаграма на модела. Избира се базов обект и се поставя на диаграмата. Следва добавяне на всички обекти, които имат връзка с него. Построяват се връзките и се определя характера им: 1:1, 1:N () , M:N (). Ето как изглежда ER диаграмата на горепосочената задача:





След построяването на ER диаграмата следва анализ на връзките. Той включва:

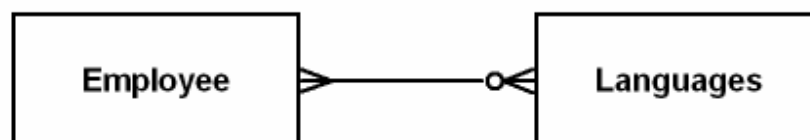
- Определяне вида (кардиналността) на връзката (1:1, 1:N, M:N).
- Определяне опционността на всеки участник във връзката – означава се с  от страната на незадължителния участник.
- Определяне на атрибутите на връзката.
- Определяне на ограниченията на връзката – надписват се под нея.

#### ✓ Кардиналност на връзката

Тя показва колко елемента от единия обект с колко от другия се свързват. Връзките от вида M:N (много към много) трябва да се преобразуват до две връзки от вид 1:M и N:1 с добавяне на нов обект между тях.

#### ✓ Незадължителност на връзката

Трябва да се определи дали връзката е задължителна. Пример за незадължителна връзка е: „служител-чужди езици, които владее” от страна на служителя, защото той може да не знае чужд език. Обърнете внимание, че за чуждият език връзката е задължителна.



Друг такъв пример е „категория на продукт-продукт”, показан на предходната диаграма.

Атрибути на връзката – трябва да се определят атрибутите, които създават връзката. Понякога се определя и времето на връзката (когато е временна). Когато

връзката има атрибути те се моделират с обект. При връзки M:N се добавят атрибути в новия обект.

✓ **Допълнителни ограничения на връзката – определяне на:**

- min и max брой записи, участващи във връзки 1:N и M:N.
- условията, които съпътстват връзката.
- предварителните условия за осъществяване на връзката.

## **Преглед на обектите**

Преглеждат се всички обекти, за да се определи:

- връзката между обектите и проблемната област;
- работните процеси, които създават, променят, използват и изграждат обектите;
- останалите обекти, които взаимодействат или зависят от текущия;
- бизнес правилата и ограниченията, които се отнасят до текущия обект;
- атрибутите на обекта.

Връзка между обект и проблемна област – по трудно се определя при връзки, моделирани с обект (привидно няма пряка връзка), например, „Един доставчик може да достави множество продукти и един продукт може да е доставен от множество доставчици”. Обектът „доставчик на продукт” моделира връзката така, както характеристиката „предпочитан доставчик” на обекта „Доставчик” за специален продукт.

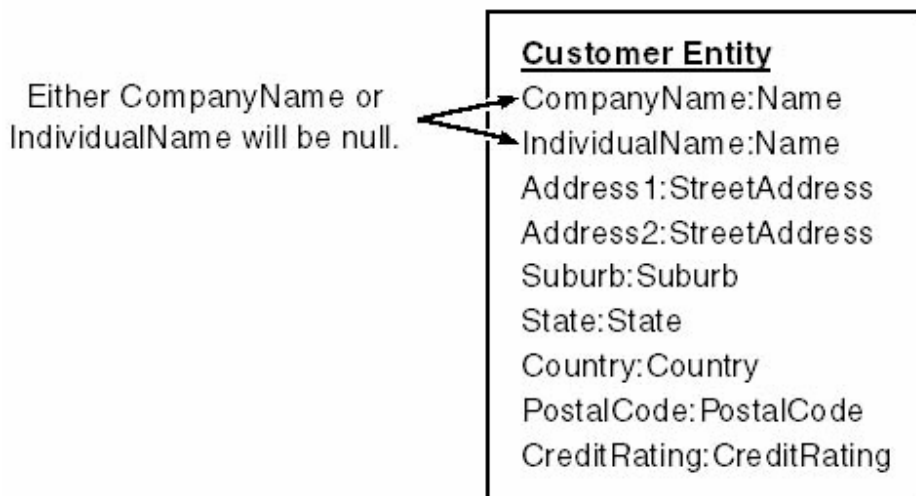
Някои понятия от проблемната област се изобразяват в един или повече обекти в модела на данните. Те се наричат съставни обекти. При работа с тях се препоръчва да се считат като прости.

**Работни процеси, които въздействат върху обект** – по-трудно се определят. Пример за такъв процес е „промяна на атрибут бонус на клиента”, който оказва влияние върху сумата за плащане. Добре е да се включат в документа такива случаи.

**Взаимодействие между обектите** – показва се чрез ER диаграма. При сложни връзки се налага допълнително описание в обекта.

**Бизнес правила и ограничения** – служи за определяне на ограниченията върху обектите и дефиниране на първичните ключове.

**Атрибути** – последната част от документацията, генерира списък от атрибути и техните области. Започва се с преглед на документите и определяне на атрибутите, след което се добавят първичните и вторичните ключове за поддържане на интегритета. Проверява се дали всеки обект има поне един кандидат ключ, който ще се използва за уникално идентифициране на всяка инстанция. Това ще бъде първичен ключ на таблицата в схемата на БД. Запомнете, че първичният ключ не може да има стойност null. Поради това изискване не винаги съществува атрибут или комбинация от такива, който да е първичен ключ. В такъв случай се добавя допълнителен атрибут, системно генериран, за да отговаря на изискванията за първичен ключ. Ето как би изглеждал обекта Customer:



Поради изискването едното от двете полета CompanyName или IndividualName да е null, нито едното от тях, нито тяхната комбинация може да бъде първичен ключ. Затова се налага добавяне на CustNo – уникален номер на клиента. Ако не съществува в организацията такова поле, ще се генерира автоматично със средствата на БД (например, AutoNumber в Access или Identity в MS SQL Server).

## Анализ на предметната област

В предходният пример списъка на атрибути е от вида име:предметна област. Много специалисти игнорират Domain и задават директно тип на атрибута и ограничение. Използването на домейн дава допълнителна информация и позволява да се дефинират ограничения в него. Освен това може да се ползва много пъти, т.е. за определяне типа на много атрибути. Ето как ще дефинираме Name от предходният пример: „низ с max дължина 75 символа, който допуска букви, точка и запетая”.

Домейн е множеството от стойности, които атрибутите могат да заемат. За да се дефинира домейн е необходимо:

- определяне на типа данни;
- задаване на ограничение в обхвата на данните;
- форматиране на домейна (незадължително).

**Избор на тип данни** – това е първата задача при дефиниране на домейн. Трябва да се определи базовият тип данни, който ще се ползва при схемата на БД. Това е единственият случай, където е добре да се разгледат съвместно концептуалният модел и схемата на БД.

**Ограничение на диапазона от стойности** – тук се определя набора от стойности, допустими за домейна. Понякога те се дават с правило като например, „Количествата са цели положителни числа”. Понякога се налага изброяване на стойности, например „Посоките могат да са една от север, юг, изток или запад”. В такъв случай домейна обикновено се включва като обект в БД и потребителя избира от списък. Единственото изключение е, когато броят на възможните стойности на атрибута е много малък и няма смисъл от обект, например „true,

false”.Обект се използва и за моделиране на домейн, задаващ стойност на повече от един атрибут. Най-добрият пример за това е домейн за щат в държава. Той зависи от страната: не всяка страна има щати, и там където има са различни.

Друга характеристика, която се посочва при определяне на домейн е може ли да има стойност null, празен низ или и двете. Полезно е дефинирането на домейн да се осъществява заедно с това на атрибут. Дефинирането на ограничения изисква точното им описание.

**Дефиниране на формати** – не са задължителни, но е добра идея на този етап да се определят подходящи формати за въвеждане/извеждане на данни. Най-подходящ пример в тази посока е работата с дати: например: „Датите да са от вида ДД-ММ-ГГГГ”.

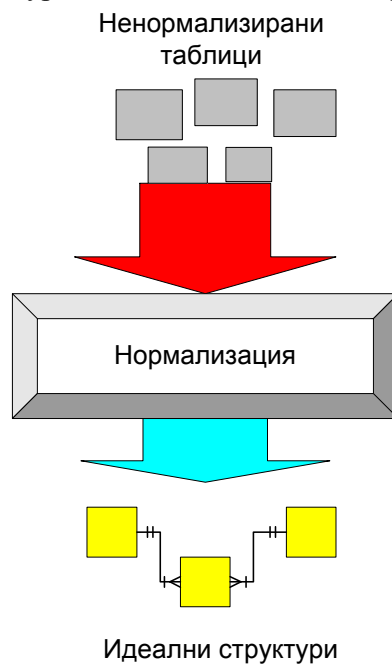
### **Задачи за самостоятелна работа:**

Да се дефинира проект на БД за:

- електронен магазин;
- склад;
- библиотека.

### **Нормализация на БД**

Следващата стъпка е нормализацията. Тя представя процес на декомпозиция на големи неефективно структурирани таблици в малки по ефективно структурирани таблици без загуба на данни. Нормализацията поддържа твърдението, че добре структурираните таблици нямат дублиране на данни и поддържа излишните данни в минимални количества (т.нар. абсолютен минимум). Това гарантира интегритета (цялостта) на данните и осигурява точност на данните при достъп до БД.



Графично представяне на процеса „Нормализация”

Процесът на нормализация означава получаване на множество нормални форми. Нормалната форма е алгоритъм, чрез който се тества структурата на таблиците. Той спомага за елиминиране на аномалиите в таблиците и полетата и спомага за получаване на ефективни таблични структури. Съществуват 7 нормални форми (NF), всяка от които се създава за решаване на специфични проблеми:

I NF	базира се на функционална зависимост
II NF	базира се на функционална зависимост
III NF	базира се на функционална зависимост
IV NF	базира се на зависимост от множество стойности
V NF	базира се на join зависимост (join обединява 2 или повече таблици)
Boyce/Codd	базира се на функционална зависимост
Domain/Key	базира се на дефиниране на домейни и ключове

### Аномалии при промяна

Причината, поради която се налага нормализация на БД, е да се избегнат аномалиите при промяна на данните в нея. Съществуват 3 вида аномалии: при добавяне, при изтриване и при актуализация.

**Аномалия при добавяне** – съществува, когато добавяне на нов запис води до безпричинни излишъци от данни. **Пример 1: данни за служители и отдели в една таблица:** не може да се добави нов отдел, ако няма поне един служител в него. Аналогично е и за служителите: не може да се добави нов служител, ако не са ясни данните за отдела, в който работи. **Пример 2: данни за клиенти и продавачи в една таблица.**

**Задача:** Представете аномалията от пример 1 с примерна таблица с данни. Обяснете аномалията в пример 2. Дайте примерна таблица с данни за подкрепа на вашето твърдение.

**Аномалия при изтриване** – съществува, когато изтриването на запис премахва данни, които не са избрани за изтриване. Отново можем да разгледаме пример 1. Тук не могат да се изтрият данни за служител, без да се изтрият данните за отдела, в който работи. А ако няма друг служител, работещ в този отдел, данните за него се губят. Наличието на аномалия при добавяне е свързано с наличие на такава и при изтриване.

**Задача:** Потърсете аномалия при изтриване в пример 2. Покажете я с примерни данни.

**Аномалия при актуализация** – тя съществува, когато промяна на определена стойност изисква същата промяна и в други записи или таблици. **Пример 3: поръчки на клиент.** В една таблица се съдържат No на поръчката, дата на поръчката, име на клиента, телефон на клиента. Ако се наложи промяна на името на клиента и за него има създадени повече от един записа, трябва да се коригират всички. Ако броят на записите е голям, операцията е бавна. А ако името някъде е сгрешено? То няма да бъде открито при търсене и актуализация.

## Зависимост: добра, лоша и застрашителна

Зависимостта попада в Теория на зависимостта. Тя е поле от науката, изучаващо Теорията на нормализацията, принципите на зависимост и други, свързани с тях теми. Повечето нормални форми се базират на различни типове зависимост и затова трябва да се познават. Съществуват 4 типа зависимост: функционална, транзитивна, множествена и join (свързваща).

**Функционална зависимост** (добра) – съществува между 2 полета A и B, когато стойност на A е директно свързана със стойност на B. Задавайки стойност на A винаги можем да определим свързаната с нея стойност на B. Означава се с  $A \rightarrow B$  и се чете „A определя B” или „B функционално зависи от A”. Добре структурираните таблици винаги съдържат функционална зависимост. Пример за нея е първичен ключ.

**Задача:** Дефинирайте таблицата клиенти (от предходното упражнение) с функционална зависимост.

**Транзитивна зависимост** (лоша) – ако за дадени 3 полета A, B и C съществуват функционалните зависимости  $A \rightarrow B$  и  $B \rightarrow C$ , то е налице транзитивна зависимост между A и C, защото стойностите на A са индиректно свързани със стойностите на C. Означава се с  $A \Rightarrow C$  и се чете „A транзитивно определя C” или „C е транзитивно зависим от A”. Добре структурираните таблици не съдържат транзитивни зависимости. **Пример 4: данни за служители и отдели в една таблица** съдържат транзитивна зависимост No служител  $\Rightarrow$  име отдел. (Полетата са: No служител, име служител, град, телефон, отдел No, име отдел.)

**Множествена зависимост** (застрашителна) – между 2 полета A и B съществува множествена зависимост когато дадена стойност на A е директно свързана с две или повече различни стойности на B. Означава се с  $A \twoheadrightarrow B$  и се чете „A определя множество от стойности на B” или „Множество стойности на B са функционално зависими от стойност в A”. Множествената зависимост може да съществува на ниво поле или запис. Подобна е на транзитивната, защото наличността ѝ показва наличие на две или повече теми в една таблица. **Пример 5: данни за служители и комисии в една таблица.** Полетата в таблицата са: No служител, име служител, телефон, име комисия, в която служителя участва. Множествена зависимост има при No служител  $\twoheadrightarrow$  име комисия, когато един служител може да е член на повече от една комисии. Това е пример за множествена зависимост на ниво поле.

**Задача:** Покажете множествена зависимост в таблица, съдържаща информация за служители и чужди езици, които владеят.

**Пример 6 за множествена зависимост на ниво запис:** таблицата от пример 5, но с уговорката, че няма множество стойности в едно поле, а за всяка негова нова стойност се добавя нов запис.

**Задача:** Покажете множествена зависимост на ниво запис в таблица служители-чужди езици, които владеят.

**Пример 7 за таблица с независими множествени зависимости:** таблица със следната структура: No служител, име служител, език, курс има множествени зависимости No служител  $\twoheadrightarrow$  език и No служител  $\twoheadrightarrow$  курс. Множествените зависимости са независими, защото едната не определя стойността на другата.

**Свързана зависимост** (join) – тя съществува в таблица A, ако всеки запис в таблицата може да се създаде чрез SQL JOIN команда, който присъединява всички

таблицы, създадени чрез декомпозиция. Това твърдение трябва да е истина за всички записи, съществуващи в таблица А по време на декомпозицията и за всеки валиден запис, който може да се въведе преди декомпозицията. Допълнително не трябва да има загуба на записи и не трябва да се добавят фалшиви записи.

**Пример 8:** таблица със структура: No продавач, име продавач, % отстъпка, статус, град продажба, телефон, Web страница. В тази таблица съществува свързана зависимост, защото може да се раздели на 2 по-малки таблици:

- No продавач, име продавач, град продажба, телефон, Web страница
- No продавач, % отстъпка, статус

Обобщената информация може да бъде получена с помощта на SQL заявка от вида:

```
select t1.id,name,discount,status,city, telNo,webpage from t1 inner join t2 on t1.id=t2.id
```

Няма изискване всички таблици да съдържат свързана зависимост. Таблицата, която я съдържа, може да бъде декомпозирана на по-малки. След декомпозицията трябва да бъде валидна свързаната зависимост. В противен случай се прилага нормализация за нейното определяне.

## Нормални форми

Преди да се стартира процеса на нормализация, трябва да се има предвид:

1. Всяка таблица трябва да има първичен ключ.
2. Таблиците не трябва да съдържат повтарящи се групи от данни.

Всяка таблица трябва да отговаря на тези условия, за да бъде процеса на нормализация ефективен.

**Пример 9:** пример за ненормализирана таблица: таблицата orders имаща следната структура id, custid, orderdate, items, където items съдържа множество стойности и представя поръчаните стоки (име стока, количество, цена и обща стойност).

**Задача:** Опишете структурата на таблицата. Попълнете я с примерни данни.

По-нататък в изложението ще използваме **relvar** за означение на релационна променлива. Всички дефиниции от релационната алгебра ще са дадени с удебелен шрифт, а логическата им интерпретация с наклонен. Под термина **ключ** ще се разбира **първичен ключ**.

### I нормална форма (1NF)

Една **relvar** е в 1NF, тогава и само тогава, когато във всяка стойност на **relvar** се съдържа само една стойност на всеки атрибут.

Целта на 1NF е да осигури, че *таблицата не съдържа множество стойности на атрибут или съставни атрибути и всяко поле съдържа само една стойност за запис*.

**Пример 9 (1NF):** таблицата orders преминава в 1NF на 2 стъпки:

1. Разделяме съставното поле items на части и всяка от тях е ново поле. В резултат структурата на таблицата orders има вида: id, custid, orderdate, item1, qnt1, price1, total1, item2, qnt2, price2, total2 и т.н. Броят на полетата се определя от запис с най-голям брой данни.

2. Следва премахване на повтарящите се полета и добавяне на записи с тази информация. Orders придобива следната структура: id, orderdate, custid, item, qnt, price, total.

**Задача:** Променете структурата на таблицата. Добавете новата таблица и ги попълнете с примерни данни.

### II нормална форма (2NF)

Една relvar е във 2NF, тогава и само тогава, когато всеки неключов атрибут е невъзстановимо зависим от първичния ключ.

За да се премине към 2NF, таблицата трябва да е в 1NF. 2NF осигурява, че *всяко неключово поле е във функционална зависимост от първичния ключ и таблицата не съдържа изчислими полета.*

**Пример 9 (2NF):** таблицата orders преминава в 2NF след разделянето ѝ на 2 таблици:

- orders със следната структура id, custid, orderdate
- OrderDetails, която съдържа полетата id, prodid, prodname, qnt, price, total

След преобразуването таблицата orders е в 2NF. При OrderDetails е налице транзитивна зависимост между id и prodname ( $id \rightarrow \text{prodid}$  и  $\text{prodid} \rightarrow \text{prodname}$ ) и в таблицата има изчислимо поле (total). За да бъде тя във 2NF се налага премахване на изчислимото поле и полето prodname. За описание на продуктите се въвежда нова таблица products с полета prodid и prodname.

**Задача:** Променете структурата на таблиците. Попълнете ги с примерни данни.

### III нормална форма (3NF)

Една relvar е в 3NF, тогава и само тогава, когато е във 2NF и всеки неключов атрибут е нетранзитивно зависим от първичния ключ.

При 3NF таблицата има следните характеристики:

- *Всяко поле се обновява независимо; промяната на стойността на едно поле не води до промяна на стойностите на други полета в записа.*
- *Всяко поле идентифицира специфични характеристики на темата, задаваща таблицата.*
- *Всяко неключово поле в таблицата е функционално зависимо от първичния ключ.*
- *Всяка таблица описва само една тема.*

**Пример 9 (3NF):** таблицата orders е в 3NF, а OrderDetails трябва да има сложен първичен ключ (id+prodid), защото в противен случай се получава  $id \Rightarrow \text{price}$ . Ако цената не се променя е удачно да се премести в таблицата products, тъй като тя зависи само от продукта и е негова характеристика.

**Задача:** Променете структурата на таблицата. Попълнете я с примерни данни.

### Boyce/Codd нормална форма (BCNF)

Една relvar е в BCNF, тогава и само тогава, когато само определящи характеристики са кандидат ключове.

BCNF е различна версия на 3NF и я заменя. Целта на BCNF е:



- *Да осигури, че поле, което определя стойността на всички неключови полета в таблицата е нейния кандидат ключ.*
- *Да осигури, че таблицата описва само една тема (това се имплицира от кандидат ключа).*

**BCNF** е по-силна от **3NF** в това, че допуска възможност таблицата да има повече от едно полета, кандидати за първичен ключ. Определянето на всички кандидат ключове в таблицата осигурява, че тя е проверена за транзитивна зависимост, последната е премахната, ако съществува, и няма аномалии.

Кандидат ключа е поле или група от полета, които отговарят на характеристиките на първичния ключ. Най-важната от тях е **функционална зависимост на всички неключови полета от първичния ключ**. След определянето на кандидатите се избира един от тях за първичен ключ.

**Пример 9 (BCNF):** таблицата orders има само един кандидат ключ (id), OrderDetails също има само един кандидат ключ (id+prodid). Всеки друг избор нарушава изискването за функционална зависимост или изискването за уникалност на първичния ключ.

Повечето таблици, които са в BCNF не се нуждаят от по-нататъшна нормализация. Изключение правят таблиците с множествена зависимост, които задължително минават през 4NF.

#### **IV нормална форма (4NF)**

**Relvar R** е в 4NF, тогава и само тогава, когато съществуват подмножества **A** и **B** на атрибутите на **R**, такива че ако  $A \twoheadrightarrow B$  е удовлетворено, то всички атрибути на **R** са функционално зависими от **A**.

Целта на 4NF е да осигури, че *таблиците не съдържат множествена зависимост и че таблицата описва само една тема. Таблицы, които съдържат множествена зависимост описват 2 или повече теми.*

**Пример 10 (4NF):** таблица EmployeeCommittee от пример 5, описваща данни за служителите и комисиите, в които участват има следните полета: id, name, tel, committee (данните в него са изброени). Налице е множествена зависимост между id и committee. За да я премахнем можем да дублираме записи, но тогава се получава множествена зависимост на ниво запис. Единственият начин да се премахне тя е с въвеждане на нова таблица с полета id и committee.

Ако има таблица с 2 множествени зависимости с независими полета се добавят 2 нови таблици и т.н.

**Задача:** За таблицата „служители-чужди езици, които владеят”, получите 4NF. Аргументирайте се с примерни данни.

#### **V нормална форма (5NF)**

Една relvar **R** е в 5NF проекция/съединение NF, тогава и само тогава, когато всяко нетривиална join зависимост, която съдържа **R**, се имплицира от кандидат ключовете на **R**.

Таблицы с 5NF *нямат транзитивни и множествени зависимости*. В повечето случаи не е необходимо декомпозиране на таблиците. Ако се появи такава възможност таблицата трябва да се провери за join зависимост. Съществуват 3 ключови въпроса, на които трябва да се отговори, за да се определи дали ще се декомпозира таблицата по-нататък:

1. *Мога ли да създам нова таблица, използвайки първичния ключ или кандидат ключовете като част от структурата на новата таблица?*
2. *Мога ли да създам оригиналната таблица чрез SQL JOIN команда?*
3. *Ще загубя ли записи при декомпозицията на таблицата?*

Ако отговорът на всички въпроси е „ДА”, то таблицата може да се декомпозира.

**Пример 10 (5NF):** таблица Employee описваща данни за служителите id, EGN, name, city, tel може да се декомпозира по:

- Общи данни за служителите: id, name, city, tel
- Конфиденциална информация: id, egn

**Задача:** За таблицата „служители-чужди езици, които владеят”, получите 5NF.

### Domain/Key нормална форма (DKNF)

Relvar R е в DKNF, тогава и само тогава, когато всяко ограничение върху R е логическо следствие на ограничението на областта и ограничението на ключа, приложен върху R.

DKNF е нова нормална форма и е подобна на BCNF в това, че частично се базира на първичен ключ и кандидат ключове. Освен тях тя се базира и на Domains. Домейнът има две страни: логическа и физическа. Логическата борави със стойности по подразбиране, диапазон на стойностите, задължителна ли е стойността и кога може да е null. Физическата страна се занимава с типа данни, дължината, броя знаци след десетичната точка (ако е число) и допустимите символи. За да бъде една таблица в DKNF, трябва да е изпълнено:

1. *Всяко поле да е пълно и подходящо дефинирано.*
2. *Всяко поле да представя характеристика на темата на таблицата.*
3. *Всяко неключово поле да е функционално зависимо от първичния ключ.*
4. *Всяка таблица да представя само по една тема.*

Таблица в DKNF няма транзитивни, множествени зависимости и аномалии. Ако една таблица е в DKNF, то тя е автоматично в 5NF.

**Пример 10 (DKNF):** таблица служители със следната структура: id, name, city, tel, departmentId, department. Таблицата може да се нормализира като се махне department, защото има транзитивна зависимост id=>department (id→departmentId и departmentId→department).

**Задача:** За таблицата „служители-чужди езици, които владеят”, получите DKNF.

### **Денормализация**

Тя се налага поради необходимостта от увеличение скоростта на изпълнение на заявките към БД. Ето най-често срещаните причини:

- Заявките се изпълняват бавно
- Отчетите се печатат твърде дълго
- Web страниците се зареждат бавно

Скоростта на изпълнение е субективен фактор. Препоръчва се денормализацията да е последна възможност. Вместо това да се опита:

- Подобрения в хардуера

- Оптимизация на мрежата и операционната система
- Оптимизация на сървъра за БД
- Използване на индекси
- Писане на добър стегнат код в съхранените процедури
- Писане на добре структурирани SQL заявки
- Подобрене на нормализираната структура на БД

Най-важното за денормализирана БД е, че винаги е налице проблем с интегритета на данните.

**Задача: Нормализирайте БД от предходното упражнение.**

## Схема на БД

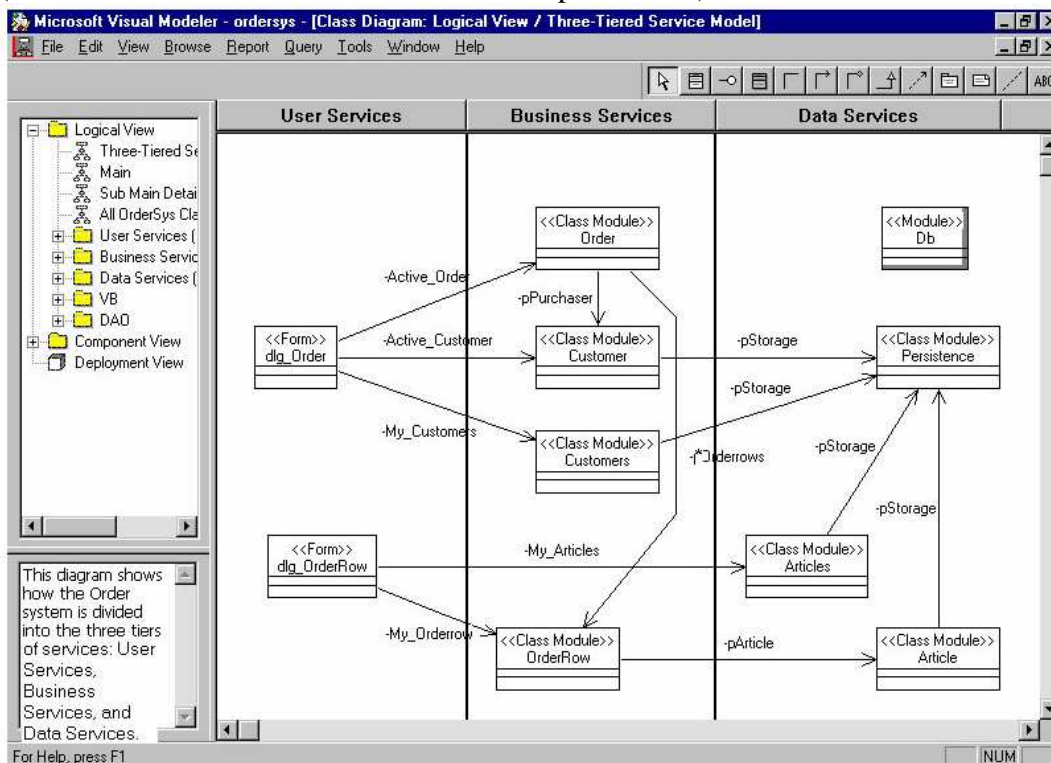
Тя описва физическата структура на данните в абстрактни термини. Действителното физическо представяне е отговорност на машината за БД (DBE). За да се опише схемата на БД е необходимо да се познава:

## Архитектура на системата

Тя представя 2 модела: архитектура на кода и архитектура на данните.

**Архитектура на кода** – често се нарича модел на приложението. Тя показва логическата структура на кода. Оказва влияние върху интегритета на данните с реализация на ограничения върху схемата на БД. Обикновено организацията на кода е на нива. Различаваме 3-слойни и 4-слойни модели – най-популярните днес.

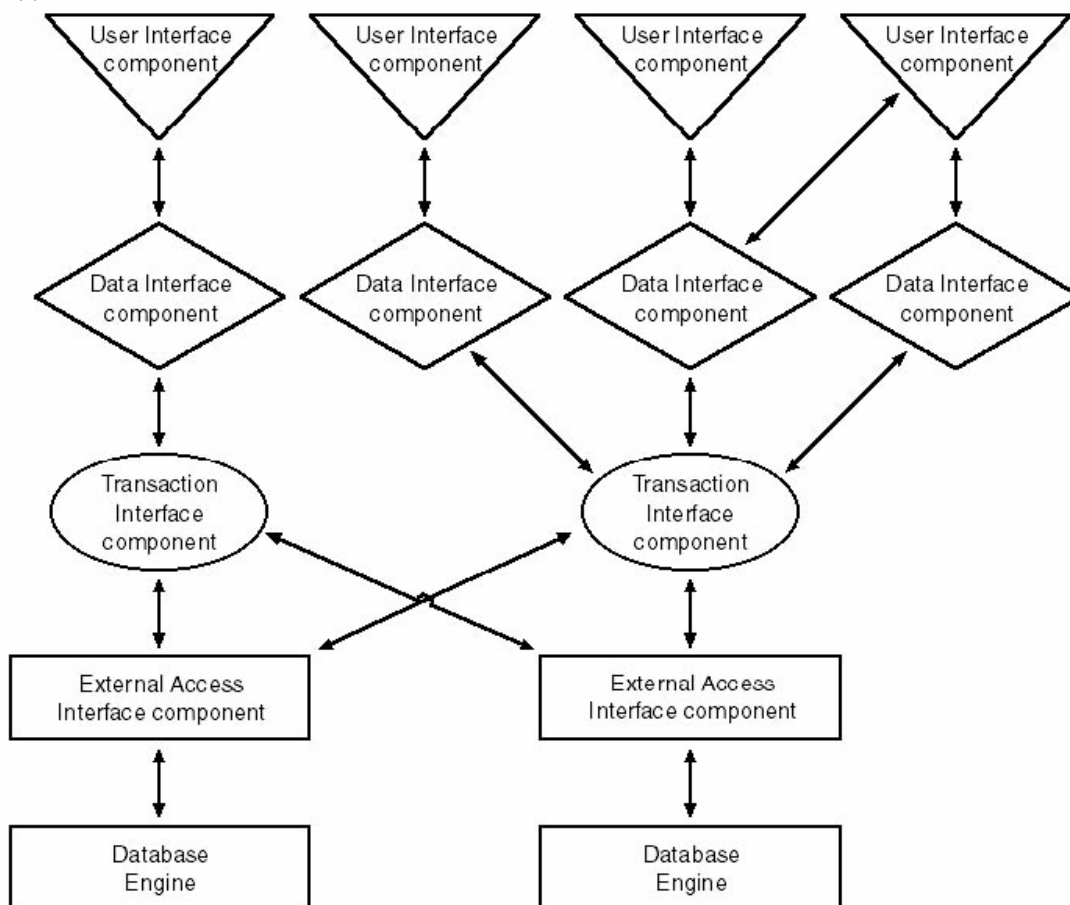
**Трислоен модел** - организира компонентите в User Services, Business Services и Data Services. Ето пример от MS Visual Modeler, част от MS Visual Studio 6.0 (MS Visual Studio Enterprise Tools):



Всички компоненти, които са отговорни за действията на потребителите и комуникират с него са в слоя User Services. Тук се намира целия GUI. Слойът Business Services е отговорен за бизнес правилата и валидиране на потребителския вход. Компонентите от този слой си взаимодействат с User Services и Data Services. Управлението на данните е поверено на Data Services, който си взаимодейства само с Business Services.

Използването на 3-слойния модел на практика създава някои затруднения. Например, форматирането на данните при печат задача на кой слой е? На Business Services или на User Services? Друг въпрос е управлението на транзакциите: кой е отговорен за тях: Business Services или Data Services? За да няма неясноти се налага уговорка във всяка конкретна задача кой слой какви правомощия има.

**Четирислоен модел** – разделянето на кода в 4 слоя вместо в 3 елиминира много от описаните по-горе проблеми. Тук слоевете са: User Interface, Data Interface, Transaction Interface и External Access Interface. Ето как изглежда 4-слойния модел на кода:



User Interface съответства на User Services в 3-слойния модел. Той отговаря за взаимодействието с потребителя, включително представя информация на потребителите във форми, отговаря на възникващите събития и инициира потребителски заявки към БД.

Data Interface управлява данните в паметта и е противоположен на External Access Interface, който се грижи за данните на диска и се намира между Transaction Interface и DBE. Data Interface явно включва по-голямата функционалност като формат на данните, създаване на виртуални таблици с резултата. В повечето случаи компонентите му са обвързани с тези от User Interface. Теоретически Data Interface компонент може да поддържа множество User Interface компоненти. Погледнато от физическа страна User Interface съответства на форма на Visual Basic (VB) или Access, а Data Interface се реализира в модула на формата. Множеството написани процедури могат да се споделят от множество форми. В този случай процедурите се намират в общи модули.

Data Interface е отговорен и за валидиране на данните, но не и за бизнес процесите. Последните са задача на Transaction Interface. Последният координира използването на данни от приложението. Компонентите му са отговорни за изграждане и инициране на заявки, получаване на информация от External Access Interface, управление на бизнес процеси и обработка на грешки и изключителни ситуации, определени от External Access Interface. Компонентите в Transaction Interface са подходящи за повторно използване от тези в Data Interface, защото се реализират от обекти на VB или Access. Пример за такъв обект е Customer, който трябва да експортира метод Update, извикван от Data Interface. За да се постигне независимост между отделните слоеве на приложението, Update трябва да работи с предаване на параметри. Той от своя страна създава SQL заявка и я предава на External Access Interface. След приключването на заявката обработва грешките от изпълнението ѝ или ги предава към User Interface.

External Access Interface е отговорен за взаимодействието между приложението и външния източник на данни. В системи с БД компонентите от това ниво се грижат за комуникация с DBE. Последният изпълнява заявки в БД и връща резултат, включително и грешки към приложението. Най-добре е на това ниво да се реализира процедура, която изолира транзакциите от спецификата на DBE. Допуска се и подмяна на External Access Interface с възможностите на Jet Engine или SQL Server.

Запомнете, че External Access Interface само изпълнява заявките, които са създадени от Transaction Interface. За да няма синтактични проблеми е добре заявките да се включват в схемата на БД. Тъй като не могат да се предвидят всички заявки, се налагат промени в Transaction Interface. При създаването на компонентите от Transaction Interface трябва да се отчете и наличието на собствен SQL синтаксис: или да е разработен само за конкретна BDE, или да има case за различни BDE.

## **Архитектура на кода и схемата на БД**

Архитектурата на кода изпълнява 2 задачи в схемата на БД:

1. Изолация на External Access Interface (или Data Services в 3-слойния модел).
2. Валидиране на данните.

За изолация на External Access Interface от промените в DBE предварително се определят необходимите заявки и се включват в схемата на БД. Някои дизайнери на БД включват и валидирането на данни в DBE. Предимствата на този подход са:

- Всички ограничения върху данните и бизнес правилата са реализирани на едно място, което улеснява актуализацията им.

Недостатъците на това решение са:

1. Множество правила не могат да бъдат реализирани на ниво DBE. Например, ако няма тригери в БД, не могат да се създадат уникални стойности на първичния ключ (Така е в Jet.)
2. Намалява се използването на системата, ако се изчаква изпращането на данни до DBE, за да се валидират. Общо правило е данните да се валидират колкото се може по-скоро след въвеждането им. Ако трябва да се спазва това правило, ще е налице интензивен обмен с DBE, което води до забавяне дори при самостоятелно приложение. Ако се наруши правилото, то потребителят ще получи информация за това как е въвел данните след като въведе целия запис, т.е. реакцията на системата е бавна.

Най-доброто решение за валидиране на данните е то да става в приложението. От друга страна ако потребителят разполага с Access или SQL Enterprise Manager, ще има контрол върху данните без валидиране. За тази цел е необходимо да се реализира проверка върху данните и в схемата на БД. Access (след версия 2000) реализира тази проверка автоматично. Когато се дефинира validation rule за поле на ниво таблица и след това се драгва полето във формата, последната наследява правилото. Ако полето се ползва м множество форми в едно или повече приложения, всички промени по правилото се правят на ръка. За преодоляване на този проблем доста дизайнери на БД правят валидацията в схемата на БД по време на изпълнение. Това не е удобно при честа промяна във валидиращите правила. Ако се използва валидация по време на изпълнение, правилата могат да се получат от схемата на БД при стартиране на приложението, при зареждане на формата или преди актуализация на всеки запис. Най-добре е да се ползва втория вариант. Първият вариант може да доведе до зареждане и изпълнение на правила, които потребителят няма да ползва. Третият вариант води до намаляване на бързодействието. Той се препоръчва само ако правилата могат да се променят динамично по време на изпълнение на приложението.

## **Архитектура на данните**

Една система за БД се състои от следните компоненти: приложение, DBE и БД. Този модел може да бъде обогатен с 3-4 слойната структура на кода(фиг.1). Теоретически всеки слой може да се намира на различна машина. Всяка логическа група от компоненти в архитектурата на данните е разположена на отделен слой. В практиката са познати следните архитектури:

### **Еднослойна архитектура**

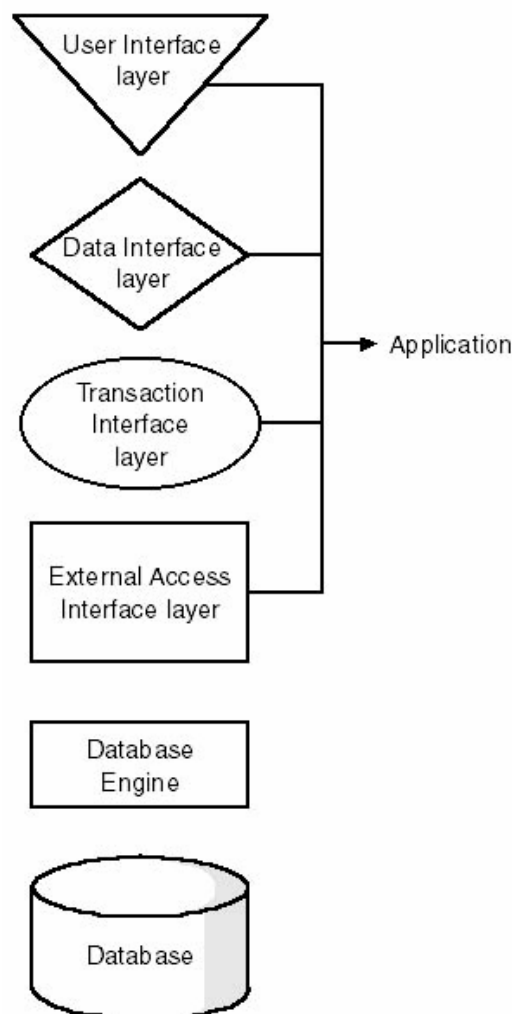
Всички компоненти се намират в един логически слой. Най-простият пример е самостоятелно приложение. При него всички компоненти са на една машина и са достъпни само за потребители, работещи физически на същата машина. Дори машината да е вързана в мрежа или в Internet, другите потребители нямат достъп до БД. Всички процеси са локални, данните също и скоростта на изпълнение зависи само от хардуера на машината (процесор и памет). Такива системи са паметоемки.

Най-използваната DBE тук е Jet машина. Съществува и MSDE, който се разпространява с Access2000 и е SQL Server Light за еднослойни архитектури.

Често срещана модификация на еднослойната архитектура е мрежова БД. При нея БД може да се намира на друга машина и се достъпва мрежово, но всички операции са локални. Приложението трябва да бъде локално защото се затруднява мрежовия трафик.

Мрежовата БД е възможна при използването на Jet машина, а не SQL Server. Тя позволява достъп на множество потребители до данните. Теоретично максималният брой потребители при Jet машина е 255, практически този брой зависи от възможностите на машината.

Намаляване на натоварването на мрежата е най-пряката ползва от схема на БД. Използването на мрежова БД предоставя достъп до отдалечен диск. Времето за достъп ще е по-голямо отколкото при локална база. Налице са 2 аспекта на схемата на БД, които директно влияят върху мрежовите характеристики: местоположение на обектите на БД и подходящо използване на индекси. Добре е интерфейсите обекти да са локални и данните, които не се променят често също да са такива. Пример за такива данни са имената на градове, държави, щати и т.н.



Фиг. 1. Архитектура на данните

Използването на индекси спестява сортировката и ускорява достъпа при търсене и подреждане. Jet машината сортира само по индексен файл (фиг. 2). SQL Server притежава специфични индекси, наречени клъстерни (clustered index), които контролират физическото подреждане на данните. Клъстерният индекс може да бъде само един на таблица.

Предимството от използването на индекси се забелязва при голям брой много дълги записи. Jet машината работи с индекс, ако е възможно. Ако трябва да се търси запис по зададено условие, търсенето в подреден индексен файл е по-бързо. Като се има предвид и това, че трябва да се преточи файла по мрежата (говорим за еднослойно приложение) разликата става значителна.

Работата с индекси трябва да отчита и факта, че при всяка промяна в БД се налага актуализация и на индексите.



Фиг.2. Индексиране при Jet машина

### Двуслойна архитектура

При двуслойната архитектура БД и DBE се намират на отделена машина. Те могат да бъдат на една и съща машина или на различни. БД може да бъде разположена на няколко физически машини, но логически системата е двуслойна. Тази архитектура съществува само на SQL Server или Oracle.

На пръв поглед няма разлика между двуслойна архитектура и мрежова БД при еднослойна архитектура. Това не е така, защото при двуслойна архитектура всички заявки за работа с БД се изпълняват на машината с БД, докато при еднослойна архитектура през мрежата се прехвърлят таблици с данни и обработките са локални (на машината на приложението). Поради тази специфика на работа често двуслойната архитектура се нарича клиент/сървър.

**Пример 11:** дадена е SQL команда **select from Customers where custId="Jonson"**. Как ще се изпълни тя? При мрежова БД Jet машината ще прочете индекса, ако съществува, ще потърси подходящ запис и ще го върне. При клиент/сървър приложение ще се изпрати заявка до SQL Server и ще се получи записа. Разликата в бързодействието проличава при сложни заявки и голям брой потребители. Подобрието се дължи и на многозадачната работа, която се ползва от ОС и БД: вместо да изчаква освобождаване на ресурс, БД или ОС може да изпълни друга задача или заявка.



Ако връзката към SQL Server става през Access, трябва внимателно да се ползват командите, които се изпълняват локално. Например, **select** с потребителски дефинирани функции ще се изпълни от Access, вместо да се предаде към сървър, защото SQL Server не поддържа функции в **select**.

### **N-слойна архитектура**

Разпространение на натоварването на процесите между две системи с двуслойна архитектура може значително да подобри натоварването на машините и оттам времето за отговор да се намали. Това означава Transaction Interface и External Access Interface компоненти да се разпределят между допълнителни междинни системи. Този подход е сложен за реализация. Много по-сложна е връзката, защитата и управлението на процесите в многослойна архитектура. Последната изисква и множество сървъри от типа на Microsoft Transaction Server. Такава архитектура се нарича N-слойна.

Освен проблеми от естество реализация налице са и проблеми по дизайна на БД. Той трудно се променя при N-слойна архитектура.

### **Internet и Intranet архитектура**

Разпространението на БД през Internet или Intranet е специален вид N-слойна архитектура. Спецификата произхожда от различната технология – HTTP и различен GUI – през browser. Логически двете архитектури са подобни.

Най-важната разлика между БД за Internet (Intranet) и традиционната архитектура е липсата на състояния (stateless) при първата. При клиент/сървър архитектура още от първия екран се изисква потребителско име и парола за връзка с БД. След получаване на връзката обикновено тя остава жива до края на сесията. При това БД знае кой е клиента и какви заявки прави, за да може да му отговори. Това познание (кой стои отсреща) се нарича **състояние** и се управлява от БД.

Когато БД се използва през Internet, сървърът няма информация за състоянието. Всеки път, когато приложението прави заявка към БД се идентифицира потребителя. След приключване на изпълнението на заявката се приключва и връзката с БД. Липсата на състояние оказва влияние върху схемата на БД и приложението. Повечето Internet приложения използват тънък клиент (thin client). Това означава, че приложението прави минимален брой обработки на клиентската страна, обикновено свързани само с GUI. Когато трябва да се показват данни, разположени на повече от 1 страници на екрана, се налага кеширане. Последното трябва да се прави в клиента, защото в БД няма информация коя страница от резултата се визуализира в момента. Ако се направи кеширане на клиентската страна, компонентите от слоеве Transaction Interface и External Access Interface трябва също да се копират при клиента. За да се избегне този процес се използва ADO. Той предоставя механизъм, наречен **paging**, за тази цел. Преди изпращане на заявката, приложението задава броя записи, които ще се върнат едновременно чрез свойството **PageSize** на обекта **RecordSet**. Свойството **AbsolutePage** определя коя страница да се върне, а **PageCount** дава броя страници.

От страна на сървъра paging води до повторно изпълнение на заявката всеки път, когато потребителят поиска страница. Това води до забавяне при сложни

заявки с много изчисления или върху голям обем данни. Когато заявката е сложна, могат да се ползват:

- Оптимизация на заявката: създаване на временна таблица, денормализация.
- Създаване на дебел клиент (fat client) вместо тънък като данните компоненти се местят на клиентската страна. По този начин може да се кешират данни на клиентската страна.

Общо правило е създаване на дебел клиент да става само при Intranet приложения, а не при Internet такива.

### **Компоненти на схемата на БД**

След като е завършен концептуалният модел и е избрана архитектурата на системата, може да се пристъпи към изграждане на схемата на БД. Тя е описание на данните обекти, които са включени в БД. При избор на много сложен модел схемата включва и дефиниране на обектите, които ще се разпространяват.

Ако използваме Access за реализация, схемата ще включва и дефиниране на всички таблици, заявки и връзки. Тя няма да включва формите, отчетите и компонентите на кода, дори да са записани в mdb файл.

Ако използваме SQL Server, схемата включва дефиниране на таблиците, views, съхранени процедури и тригери.

### **Дефиниране на таблици и връзки между тях**

Дефинирането на таблици произтича от концептуалния модел. Обектите стават таблици, атрибутите са техните полета. По-голямо внимание трябва да се обърне на ограничението, връзките между таблиците и индексите.

**Ограничения** – като част от концептуалния модел на данните се дефинират ограничения за обекти, атрибути и домейни. Дали ще се реализират в схемата на БД зависи от избраната системна архитектура. Дизайнерите на БД предпочитат всички ограничения да се реализират в Data Interface и Transaction Interface слоеве при 4-слоен модел или в Business Services при 3-слойна архитектура на кода. Ако решите ограниченията да са част от БД, то те трябва да се включват в схемата. При това голяма част от ограниченията върху области и атрибути стават ограничения на ниво поле в схемата и в Access се представят чрез validation rule. Access поддържа и CHECK клауза, използвана от SQL Server, което може да се използва при изграждане на БД със SQL, а не с DAO или с интерфейса на Access.

Ограниченията на ниво обект става ограничения на ниво таблица и се реализират по същия начин. За запазване на интегритета на БД се ползва първичен ключ, автоматично генериран, за да е уникален.

При реализацията на БД чрез SQL Server или Jet машина ще забележите, че не всяко ограничение може да се реализира в дефиницията на таблицата. В SQL Server могат да се ползват тригери за тази цел, а при Jet машината, където няма тригери, се залагат като част от приложението.

**Връзки** – първата стъпка е включването на уникален идентификатор от първичния ключ към външната релация. На ниво схема на БД това означава, че първичният ключ от първичната таблица ще се включи във външната таблица.

Някои дизайнери спират дотук, предпочитайки да използват интегритета на референциите на ниво приложение, а не в схемата на БД. Най-добре е да се ползват и двете техники.

**Индекси** – всяка таблица трябва да има поне един индекс, който DBE създава автоматично при декларация на първичния ключ. В добавка трябва да се създаде индекс за всяко поле или комбинация от полета, които се ползват в **join**. Трябва да се декларират и допълнителни индекси в таблицата, представящи външни релации ако полето (полетата) не съвпадат с целия първичен ключ на външната релация. В този случай се дефинират отделни индекси върху външния ключ.

Всички полета, по които ще се сортира също се индексират. Например, списъка с потребители може да се подреди по име, по номер. Така сортирането става по ефективно.

Трябва да се внимава да не се създават голям брой индекси. Затова вместо индекс, може да се използва **Order By** клауза. Максималният брой индекси зависи от това, колко често се актуализира таблицата. Например, таблица за клиентски поръчки не трябва да има повече от 10-15 индекса.

## Views и queries

И Access, и SQL Server дават възможност за съхранение на **select** заявка. Така съхранения оператор се нарича **view** в SQL Server и **query** в Access. Използването му повишава бързодействието. Определянето кои заявки ще се запишат в БД се базира на концептуалния модел като се избират сложни заявки. Включването на заявка в схемата на БД денормализира сложните обекти. Такива обекти включват много таблици с връзки 1:1 и 1:N.

Трябва да се прегледат и всички заявки във формите и отчетите на приложението. Заявките са необходими и за връзки с полета от тип **lookup**. Допуска се включването и на заявки, които изпълняват действия. За разлика от индексите, броят на съхранените заявки и съхранени процедури се препоръчва да е максимално голям. Те са начин за ускоряване на изпълнението и не водят до претоварване на БД.

Запомнете, че разработката на системи с БД не е линеен процес. Докато промените по таблиците по време на реализация могат да предизвикат проблеми, то добавянето на заявки към схемата на БД е тривиално и се очаква.

## Сигурност

Следващият етап от проектирането на БД е определяне на административните изисквания на системата. Това са метаизисквания, които се отнасят до системата като цяло, а не до предметната област, моделирана от нея. Те се делят на 2 категории:

- Изисквания от защита – определят кой има достъп до системата.
- Изисквания за достъп – определят колко често системата трябва да е on-line и как потребителят ще архивира данните.

Тъй като достъпът се дефинира по време на реализация, тук ще се спрем върху защитата. Реализацията на защита е сложен, комплексен бизнес. Тук ще представим логическото ниво на защита и принципите му. Конкретната реализация зависи от БД.

## **Нива на защита**

Това е първото, което се определя. Говорим за защита на данните, а не на кода. Най-ниското ниво е незащитена система, в която всеки има достъп до БД по всяко време. Това ниво се реализира и администрира много лесно, защото не се налага използването на допълнителни средства.

Следващото ниво е share-level. При него се задава парола на цялата БД и всеки, който я знае има пълен достъп до системата. Тази защита е достатъчна в много случаи.

User-level е защита, която изисква повече усилия за реализация и администриране. Тя дава най-голям контрол върху БД. Тази защита позволява на системния администратор да дава специфични привилегии на потребителя върху всеки обект. При нея се определя вида на операцията върху БД и кой има право да я изпълнява. Привилегиите могат да се раздават както на индивидуални потребители, така и на групи, към които потребителят принадлежи. Наличието на групи прави защита по ефективна, защото изисква по-малко администриране. Използването на този модел изисква първо идентифициране на типа потребител, а след това се раздават права за всеки обект в системата. Не е необходимо да се присвояват права на обектите за данни. Привилегиите касаят формите и функционалността по тях.

Често се налага да се забрани достъпа до порция данни, например ЕГН, за част от потребителите. В този случай трябва да се зададат права върху заявката и да се отнеме достъпа до полетата и таблиците.

## **Проверка**

В добавка на контрола кой има достъп до данните, се налага и да се знае кой потребител какво прави във системата. Тези изисквания могат да варират. Някои организации искат да знаят кой е влязъл в системата и кога. Други искат детайлна проверка кой какви промени е направил, трети се намират по средата.

Моделирането на тази проверка зависи от изискванията. Ако трябва само да се следи кой кога е влязъл, се дефинира обект с атрибути UserName, LogOn и LogOff. При влизане в системата се добавя запис към този обект, а при излизане се актуализира LogOff. Понякога се налага да се разбере кой е добавил запис. Тогава към обекта се добавят 2 атрибута: CreatedBy и CreatedOn.

По-сложен е въпросът с изтриването. Една от възможностите е да не се изтрива физически, а да има флаг, който маркира записите за изтрити. Освен това към обекта се добавят атрибутите DeletedBy и DeletedOn. Тази техника е подходяща и ако трябва да се архивират записите преди изтриване от БД. Друга възможност е изтриване на записа и запис в log file, по същият начин, по който се прави запис при влизане в системата (виж по-горе).

Ако трябва да се знае какви промени са направени е необходимо да се добави допълнителна таблица за проверка към модела за всеки обект. Тя ще съдържа

информация за потребителя, който е направил промяната, датата и вида на промяната, Но на записа, който е променен, старата и новата стойности.

Ако ще реализирате тези проверки с Jet машина, трябва да забраните на потребителите да имат директен достъп до таблиците, защото той ще им позволи да заобиколят защитата. SQL Server не се нуждае от такава забрана, защото поддържа тригери, които не могат да се пренебрегнат.

С моделирането на проверката, трябва да помислите как ще се използва информацията, от кого и при какви обстоятелства. Трябва да се ограничи достъпа до проверките. Налага се добавянето на работни процеси към дизайна на системата, с които да се управлява проверката. Трябва да се помисли и за достъпа на системния администратор до тях, както и за необходимите отчети, които да се генерират от тях. Когато създаването на тази проверка е само от съображения за сигурност и се ползва в изключителни случаи, не се налага създаване на работни процеси. Системните администратори лесно могат да генерират справка интерактивно.

### ***Задачи за самостоятелна работа:***

Дефинирайте схемата на БД за избрана от вас задача:

- електронен магазин;
- склад;
- библиотека.