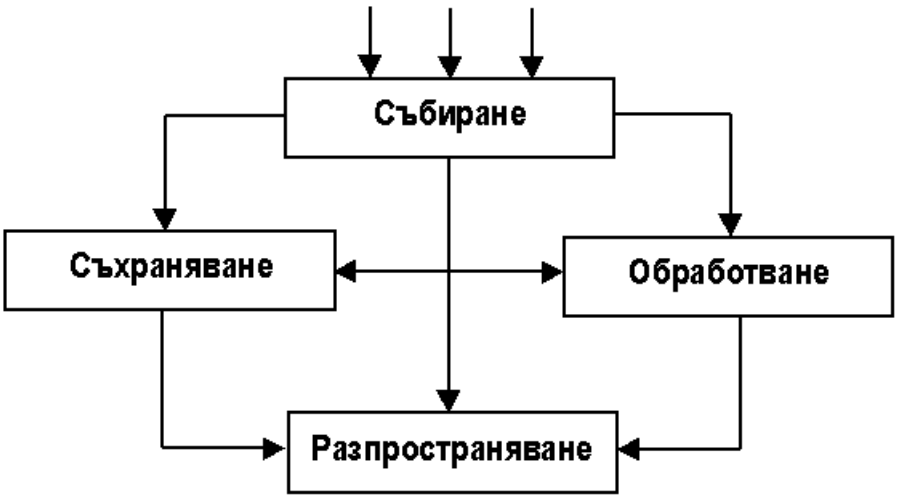
1. Въведение. Основни положения

1.1 Информация. Информационни дейности

Стремежът към получаване на знания за заобикалящата действителност е присъщ на самата природа на човека. Познанията, сведенията за реалния свят, използвани при взаимодействието ни с него ще наричаме информация. Понятието информация е многостранно. При определянето му се следват различни подходи и се оформят различни гледни точки. Философията разглежда информацията като философска категория наред с другите такива. Лингвистиката разглежда представянето на информацията чрез знакови системи. Психологията се интересува от ролята на информацията в процесите на общуване и в познавателните процеси.

В началото на 20 век е обърнато внимание на ролята на информацията в съобщенията и необходимостта от измерването и. Появила се теорията на информацията на Шенон, според която информацията е средство за намаляване на неопределеността. Въвежда се единица за измерване - бит, 1 бит е информация за изхода от събитие с два възможни изхода (1 бит е информацията за пола на даден човек). Количеството информация в дадено съобщение (според теорията на Шенон) е разликата в неопределеността преди и след получаване на дадено съобщение.

За кибернетиката информацията е необходимо да се обработи, за да се вземат управленски решения. Информатиката, която изучава информационните процеси и начините на тяхното автоматизиране, се интересува главно от структурите и формите на информацията и начините на представянето и. Такъв смисъл в понятието информация ще влагаме и ние в нашия курс.



фиг. 1.1

С информацията се извършват разнообразни дейности, които могат да се групират в четири основни групи (фиг. 1.1):

1. Събиране - търсене на подходящи източници, подбиране по определени критерии, регистриране, представяне;

2. Съхраняване - осигурява пренасяне на информацията във времето. Необходимо е тя да се подреди и структурира за лесен следващ достъп. Съществен момент е и защитаването от повреждане и загуба;

3. Обработване - получаване на информация, която не фигурира в явен вид първоначално;

4. Разпространяване - определяне на потенциалните потребители и привеждане в подходяща за тях форма.

Четирите типа информационни дейности рядко се срещат в чист вид. Произволен информационен процес представлява съчетание от всичките или от някои от тях.

1.2 Величини и данни

Величина - абстрактен информационен обект за изразяване на определени свойства на даден обект от реалния свят. Обикновено се характеризира с име и тип.
Данни - стойностите на отделните величини. Характеризират се с тип.

От определенията за информация и данни, следва че всяка информация представлява съвкупност от данни.

1.3 Файлови системи и бази от данни

Използването на компютрите дава възможност да се събират, съхраняват, обработват и използват данни по много по-удобен начин. Тези действия с данните са се извършвали винаги, но по различен начин.

1.3.1 Файлова система (File Processing System)

Всяко предприятие или организация има различни информационни нужди, които удовлетворява чрез използване на различни компютърни системи (най-често наричани информационни системи). За всеки конкретен случай се разработва съответна приложна програма или система (File Processing System). В една организация обикновено такива програми се разработват независимо една от друга, могат да използват едни и същи данни, представени и съхранени по различни начини, без да се мисли за използването им за евентуални бъдещи приложения. Приложната програма е предназначена за решаване на една конкретна задача или съвкупност от свързани задачи. Приложната система обединява няколко приложни

програми, файлове от данни и процедури, които обслужват отделна функция или процес в дадена организация.

Файловите системи притежават редица недостатъци:

- Излишъци (Redundancy) - Обикновено всяка система използва собствени файлове с данни, то едни и същи данни се съхраняват многократно (в различни приложения), като за съхраняването им се използват различни начини. Паметта за съхраняване на тези данни се използва неефективно.

- Противоречивост на данните (Inconsistent Data) - Различни файлове съдържат едни и същи данни, които в даден момент имат различни стойности. Това създава сериозни проблеми при необходимост от определяне на актуалните стойности.

- Затруднено модифициране на едни и същи данни - Тъй като различните приложения могат да използват едни същи данни, които са събирани по различно време и съхранявани по различен начин, то обновяването им е трудно, а понякога дори невъзможно.

- Ограничено използване на вече събрани данни - Тъй като всяко приложение използва свои собствени данни, представени и съхранени по собствен начин, вероятността те да могат да се използват за следващи приложения е твърде малка. Освен това информацията, използвана за едно приложение на практика е неизползваема за потребителите на други приложения.

- Невъзможност за стандартизиране на представената информация - Разработчиците на различните приложения обикновено не считат за необходимо да съхраняват информацията по начини, които са съобразени с някакви стандарти.

1.3.2 Бази от данни

База от данни (БД) може да се разглежда като съвкупност от взаимосвързани данни, организирани и представени по начин, който позволява използването им от различни потребители за удовлетворяване на различни информационни нужди на дадена организация.

Какво ново предлага използването на БД?

- Могат да бъдат избегнати излишъците от данни, а когато е необходимо наличието им, те могат да бъдат контролирани.

- Може да се осигури цялостност (integrity), непротиворечивост на данните - те да бъдат актуални и с едни същи стойности навсякъде, където се появяват.

- Информацията в БД може да бъде използвана от много потребители по едно и също време.

- Може да бъдат балансирани конфликтните ситуации при обновяване на данните.

- Може да се обезпечи сигурност на данните, да бъдат предпазени от нежелани въздействия и недопустими корекции.

- Могат да се прилагат стандарти, така че данните от една БД, даже цели файлове да се използват в други такива.

- Може да се осигури привилегировано използване на данните - части от тях да бъдат достъпни само за отделни потребители.

1.3.3 Независимост на данните

Много важна характеристика, може би най-съществената, отличаваща БД от файловата система, е независимостта на данните. Този въпрос е много съществен и изисква специално внимание.

При файловите системи всяка приложна програма работи със собствени файлове от данни и, при необходимост от промяна в тези файлове с данни, винаги е необходимо да се направят промени (обикновено не прости) в приложната програма. Организирането на данни в БД позволява това неудобство да бъде преодоляно. Данните се организират и описват чрез специализирани за тези цели средства, които не изискват описание на файловете с данни. Приложните програми, които обработват данните от БД, използват тези описания. Независимостта позволява да се извършват действия с данните без да се знае тяхното физическо разположение и без да се знае колко точно файла се използват за съхраняване на нужните данни и как точно са организирани тези файлове. Възможно е данните да могат да бъдат преобразувани винаги, когато това е нужно, да се добавят или отстраняват данни, да се променя структурата на файловете, в които те се съхраняват, да се добавят нови стандарти или да се променят съществуващите, възможно е съхраняването на съществуващите данни на друго запомнящо устройство и други подобни действия, които са недопустими или силно затруднени в обикновената файлова система.

Съществуват две нива на независимост на данните:

- логическа независимост - възможно е да се променя структурата и организацията на файловете с данни, без да е необходима промяна в приложните програми, използващи данните от тези файлове;

- физическа независимост - възможно е да се променят начинът на съхраняване на файловете върху физическите носители, достъпът до тях, типът на физическото устройство, без да са необходими промени в описанията на данните и в приложните програми, които ги използват.

1.3.4 Системи за управление на бази от данни (Database Management Systems)

За всяка БД трябва да съществува възможност за постоянни промени, свързани с промените на информационните нужди на организацията, за която тя е предназначена. Затова трябва да съществуват програмни и технически средства за добавяне, отстраняване, актуализация на данните в БД, използване на тези данни за получаване на различни справки, разширяване и реорганизация на самата БД. Тези средства, заедно с БД, представляват Система за управление на бази от данни (СУБД).

Когато говорим за СУБД, важно е да знаем, че винаги участват четири основни компоненти:

- данни - това е информацията, събрана, съхранена в БД, която може да бъде обработвана по различен начин и е достъпна за всички потребители във всеки момент;

- технически средства - това са носителите, върху които се съхраняват файловете с данни. Има машини, които разполагат със специален хардуер за работа с БД.

- програмно осигуряване - съвкупност от програми, които управляват и извършват всички действия с данните, в отговор на приетите заявки на потребителите;

- потребители - лицата, които ползват данните от БД. Те могат да бъдат групирани в три основни групи:

Администратор на БД - лице или група лица, които имат най-висок приоритет за достъп до данните от БД. Той има право да преструктурира и реорганизира БД и да организира достъпа на останалите групи потребители;

Приложни програмисти - потребители, които пишат приложни програми, за работа с данните от БД. На тях е позволено да извършват всякакви операции с данните.

Крайни потребители - лицата, които работят само с вече написани (съществуващи) приложения. Повечето СУБД предлагат специализирани езици или менюта за работа с данните. Те също се разглеждат като вградени приложения и тези, които ги ползват, също са крайни потребители.

1.3.5 Езици за работа с данните

Почти всяка СУБД предлага специализиран език, с помощта на който се описват всички действия с данните в дадена БД. Командите на тези езици обикновено се делят на две основни групи:

- Език за описание на данните - ЕОД (Data Definition Language - DDL). Включва командите за описание на данните в една БД и връзките между тях. Резултатът от обработката на тези команди се съхранява в БД и се използва при работата с данните от нея;

- Език за манипулиране с данните - ЕМД (Data Manipulation Language - DML). Включва команди, които позволяват да се описват различни действия с данните от БД - добавяне, изтриване, актуализация и получаване на различни справки.

В повечето СУБД няма точно разграничаване на двата езика. Някои системи допускат достъпът до данните да се осъществява от програми на универсални езици за програмиране, в които са вградени възможности за работа с данни от БД.

1.4 Идея за изграждане и използване на БД

Сега в различни предприятия и организации за работа с по-големи количества данни се използват както Файлови системи, така и БД. С цел ефективно използване на БД, непрекъснато са се подобрявали и развивали теоретичните основи за тяхното изграждане. Те се развиват и сега. Въпреки това, основните принципи за изграждане и използване на една БД не се променят. Те се заключават в следното:

1. Из следване на информационните нужди, които създаваната БД има за цел да удовлетворява;

2. Избор на подходяща СУБД, която ще бъде използвана;

3. Изграждане на логически модел на данните;

4. Изграждане и "напълване" на самата БД и заедно с това изграждане на нейния физически модел;

5. Създаване на приложения за разрешаване на всяка необходимост за използване на данните от БД;

6. Оформяне на някои приложения във вид на информационни системи;

7. Администриране и поддържане на БД.

2. Модели на данните

2.1 Предметна област

Всяка конкретна БД представя строго определена част от заобикалящата ни действителност, наречена предметна област. Предметната област разглеждаме като съвкупност от хора, предмети, събития (Entities), които взаимодействат помежду си, за да изпълняват определени задачи.

Промените в предметната област предизвикват съответни промени и в БД. Ето защо, предметната област може да се разглежда като източник на данни, които се съхраняват в БД. Взаимодействието на компонентите, съставящи предметната област, чрез което се постигат нейните цели, също е обект на представяне в БД.

Могат да се разграничат три нива на разглеждане на предметната област:

- част от заобикалящия ни свят (предприятие, организация...), за която се съдържат данни в БД - предметна област (Reality);
- описание на данните в БД (Metadata);
- реално, физически съществуващи данни (Data).

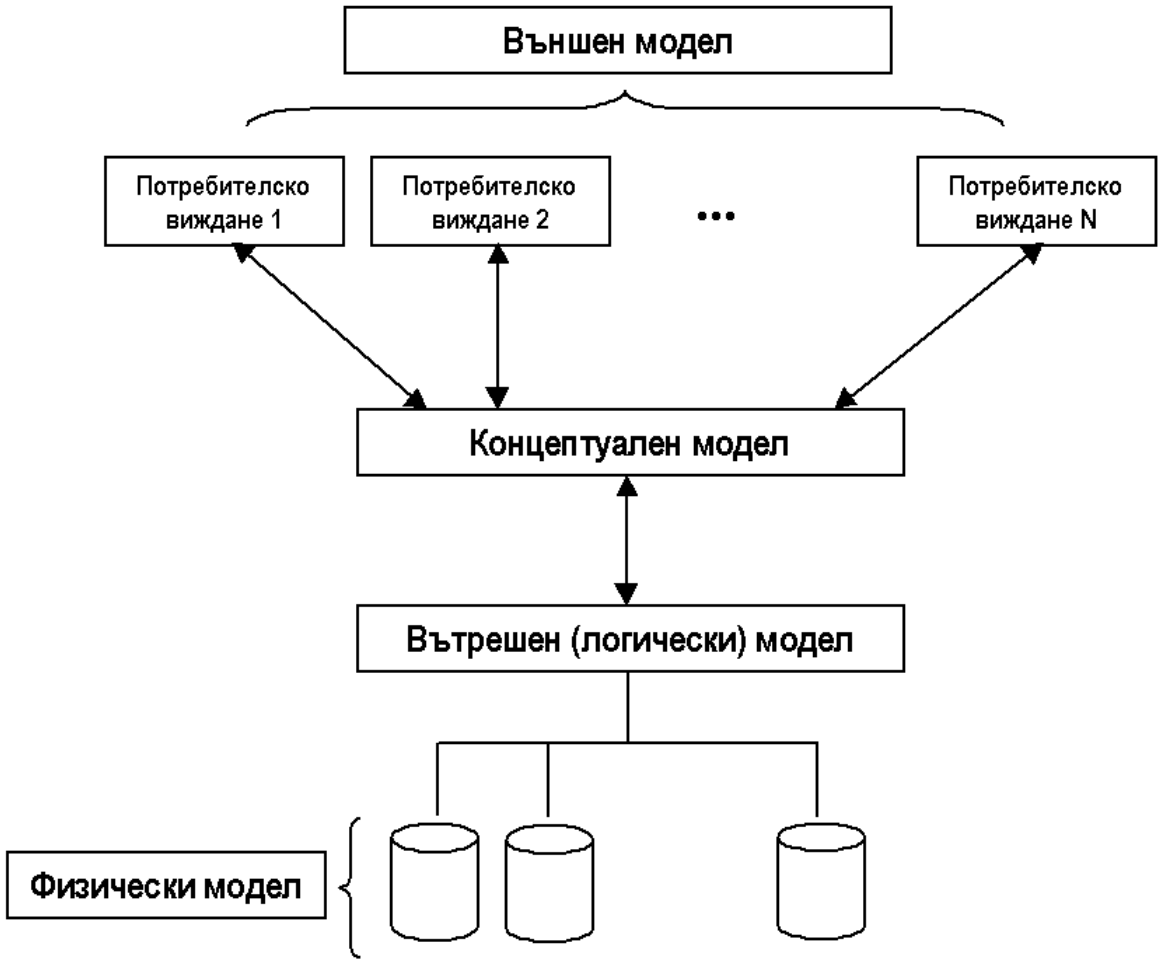
2.2 Описание на предметната област. Моделиране

Моделът на данните е абстрактно средство за представяне (описване) на обекти, събития, дейности и взаимодействието им в една предметна област. Изграждането му позволява не само да се представят данните, но и да се изяснят връзките между тях по нагледен и удобен за лесно възприемане начин. При създаване на модел на данните винаги се взема под внимание смисълът на данните, който се влага в тях като данни от конкретна предметна област.

Разглеждат се три нива на абстракция при разработване на модел на данните в една БД (фиг. 2.1): външен модел (External model or Views), концептуален модел (Conceptual model), вътрешен (логически) модел (Internal model or Schema), които реално (физически) са организирани и представени чрез физически модел.

При изграждане на всяка БД се използват представите на различните групи бъдещи потребители, за да се изяснят техните информационни нужди. Така се създава външният модел на данните, който има за цел да опише като отделни модели данните, които интересуват всеки конкретен потребите и връзките между тях.

Концептуалният модел обединява представите и изискванията на различните потребители в едно цяло като съхранява всички необходими връзки и изглажда евентуално възникналите противоречия.



фиг. 2.1

Вътрешният модел се създава като се преобразува концептуалният модел според изискванията и ограниченията на конкретна СУБД. Вътрешният модел се нарича още логически модел.

2.3 Модел "Същност - връзка" (Entity Relationship Model/ERM)

2.3.1 Същности и атрибути

Същност (Entity) - един обект, лице, явление, за което в БД се съхраняват данни, характеризиращи го като елемент именно на тази предметна област.

Клас от същности (Entity class, Entity Set, Entity Type) наричаме съвкупност от еднотипни същности - Студент, Купувач, Пациент. Всяка същност принадлежи точно на един клас от същности когато се разглежда като елемент от дадена преметна област. Обаче същото лице, предмет и т.н., като елемент на друга предметна област, може да принадлежи на съвсем друг клас от същности.

Всеки клас от същности, като елемент на дадена предметна област, се представя в БД чрез съвкупност от свойства (характеристики), описващи го като елемент именно на тази предметна област. Тези характеристики се наричат атрибути (Attributes). Стойностите на атрибутите са индивидуални характеристики на всяка същност.

Всяка същност от даден клас трябва да притежава поне един атрибут, който я отличава от другите същности в този клас. Такъв атрибут се нарича идентификатор (Identifier). Казва се, че идентификаторът функционално определя останалите атрибути за тази същност. Идентификаторът се нарича още ключов атрибут.

Различните същности от даден клас се наричат представители на класа. Те се отличават една от друга по стойностите на атрибутите си.

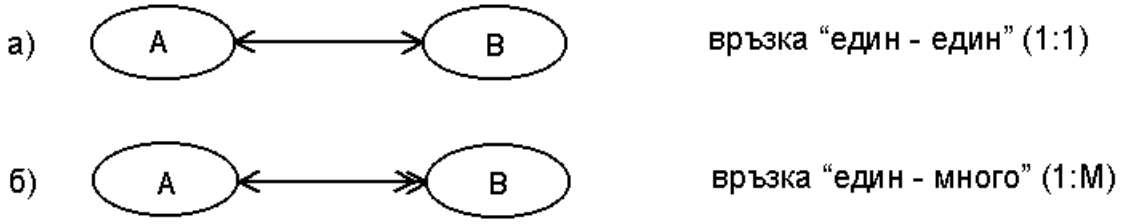
2.3.2 Връзки между класове от същности. Класификация на връзките

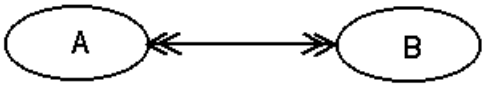
Важно свойство в една предметна област е връзка (association), която може да съществува както между две същности от един клас, така и между същности от два и повече класове. Представянето на връзки между същности е едно от съществените преимущества, които отличават БД от обикновените файлови системи.

Във всяка БД съществуват твърде голямо количество елементи на данните, между по-голяма част от които съществуват различни зависимости. Не всички, обаче, имат смисъл от гледна точка на целите, за които се създава конкретната БД. Ще се ограничим с връзките, коиото съществуват между класове от същности.

Наличието на връзка между два елемента означава, че когато е известна стойността на единия от елементите, то можем да кажем каква стойност (стойности) има другият. Връзките между два елемента могат да се разглеждат както като еднопосочни, така и като двупосочни, тъй като, обикновено, ако А има връзка с В, то и В има връзка с А.

В зависимост от това колко стойности на единия елемент са свързани с една стойност на другия елемент различаваме следните типове връзки:



в) 

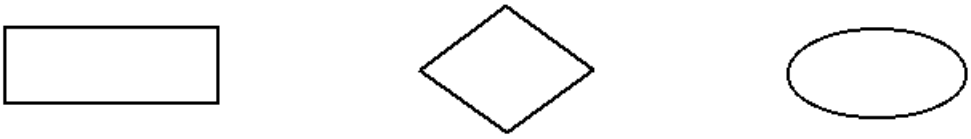
връзка “много - много” (M:N)

Типът на връзката се определя в зависимост от това калко представителя на единият клас от същности с колко представителя от другия клас се свързва.

За разработване на външния и концептуалния модел на конкретна БД най-често се използват моделът "Същност - връзка", който е удобен за този етап от проектирането на БД, тъй като е независим от изискванията на конкретна СУБД. При неговото изграждане се изследват различни варианти на моделиране на дадена предметна област като съвкупност от класове от същности, атрибути, които характеризират класовете и връзките между отделните класове.

2.3.3 Основни символи на модела

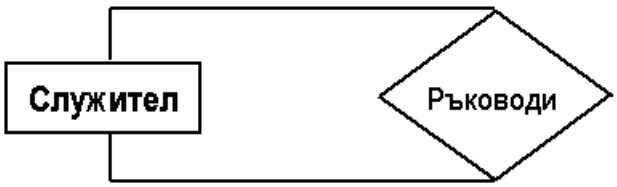
За представяне на модела “Същност – връзка “ се използва диаграма “Същност – връзка” (Entity Relationship Diagram/ERD). За създаването и се използват символи, чрез които се изобразяват отделните компоненти на модела:



Клас от същности | Връзка | Атрибут

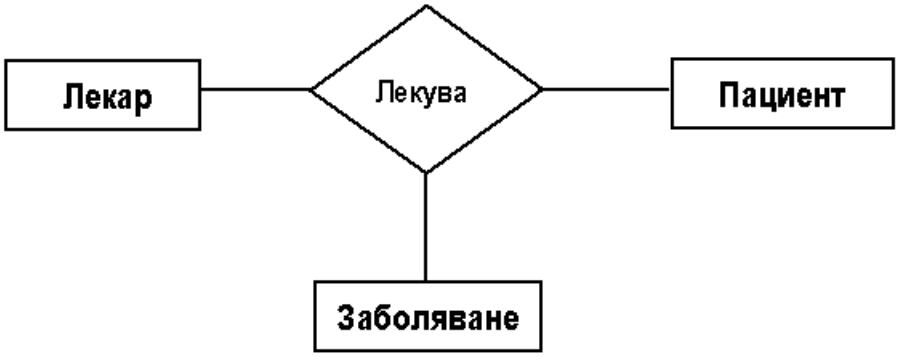
Всеки компонент на модела има название, което се разполага в съответния символ. Например:

Връзките, дефинирани между отделните класове от същности, се характеризират със степен, която се определя от броя на класовете, участващи във връзката.



а) Унарна връзка

б) Бинарна връзка



в) Връзка от трета степен

Връзки от степен по-висока от трета се срещат твърде рядко. Най-често използваните връзки са бинарните.

Всички елементи на ERM (класове от същности, атрибути, връзки) се именоват, като за връзките се посочва и техния тип. Някои връзки могат да имат атрибути, които ги характеризират. Такива са връзките от тип “много - много”. Като атрибути на връзките в този случай се обособяват такива атрибути, които са общи за класовете, свързани с дадената връзка.

2.3.4 Пример за разработване на модел “Същност – връзка”

При разработване на този модел е необходимо да се дефинира следната информация за предметната област:

1. Списък от класове от същности, съставящи предметната област.
2. Списъци от атрибути на всеки клас от същности.
3. Връзки, които съществуват между някои от класовете, тип и степен на връзките и евентуални техни атрибути.

Моделът “Същност – връзка е удобен с това, че процесът на определяне на класовете от същности, атрибутите и връзките е итеративен. След разработване на първи, приближен вариант, се извършва доуточняване, което води до корекции в модела.

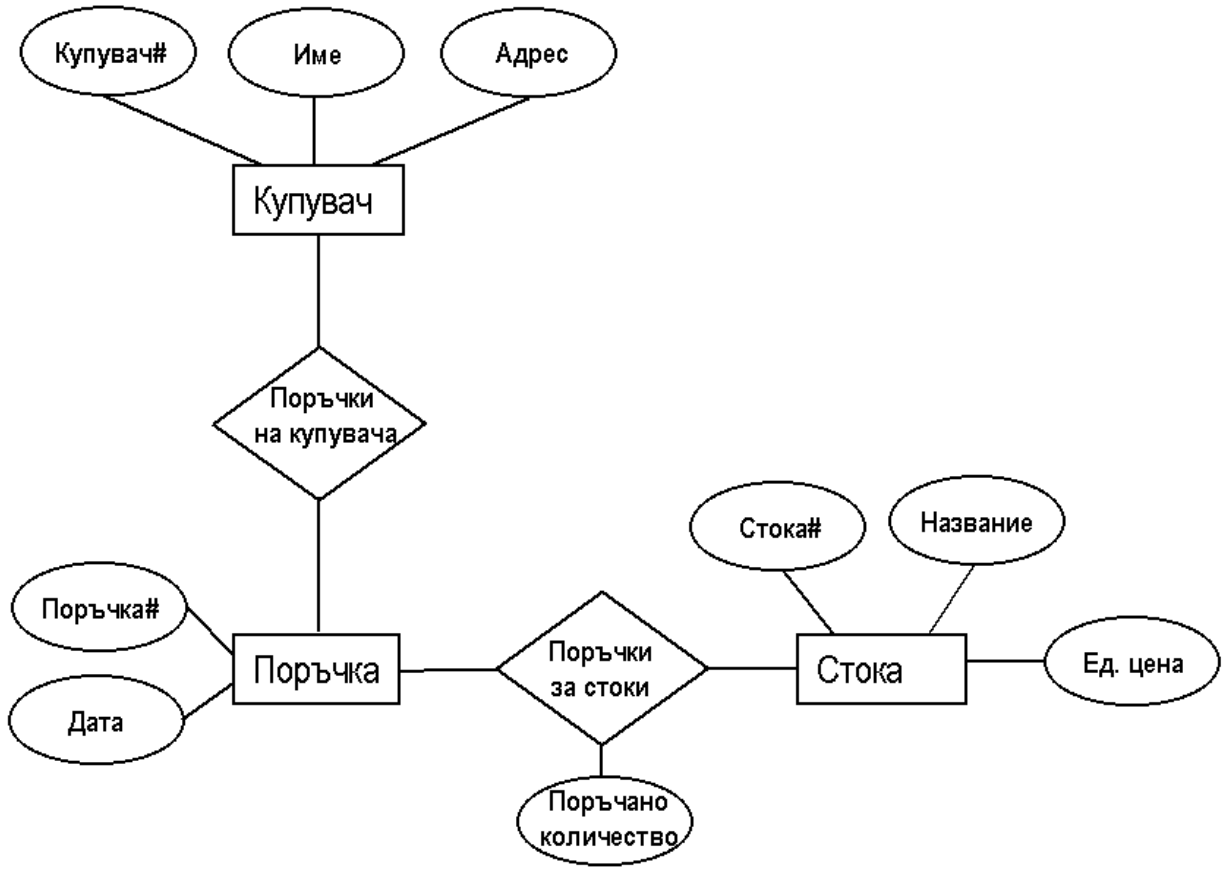
Да предположим, че е необходимо да разработим БД за магазин за търговия по каталог, в която е необходимо да се съхранява информация за купувачите, стоките, които се предлагат в магазина и поръчките, направени от купувачите.

Ще определим клас ***Купувач***, който се характеризира със следните атрибути: ***Купувач#, Име*** и ***Адрес***. Необходим е и клас ***Стока***, който се характеризира с ***Название, Стока#*** и ***Единична цена***. Купувачите правят своите

заявки за покупка чрез попълване на поръчки, които могат да се представят посредством клас ***Поръчка*** с атрибути ***Поръчка#*** и ***Дата***. Удачно е да определим връзка между класовете ***Купувач*** и ***Поръчка***, която показва кой купувач какви поръчки е направил. Друга естествена връзка е тази между класовете ***Поръчка*** и ***Стока***, която отразява какви стоки са включени в дадена поръчка. Ако е необходимо да включим информация за това от коя стока какво количество е поръчано в дадена поръчка, то можем да добавим атрибут ***Поръчано количество*** на връзката между класовете ***Поръчка*** и ***Стока***.

Тези две връзки са достатъчни и чрез тях неявно можем да установим връзка и между различни представители на класовете ***Купувач*** и ***Стока***, за да получим информация кой купувач какви стоки е поръчал.

В резултат на направените разсъждения ще получим следния модел:



2.4 Вътрешен модел на данните

Различните СУБД предлагат основно три модела за представяне на данните: йерархичен, мрежов и релационен. Тези модели са се появили последователно с развитието на теоретичните основи на БД и възможностите за тяхната практическа реализация. Изборът на вътрешния модел е важна задача при проектирането на БД и се прави както в съответствие с изискванията на

СУБД, така и с отчитане на спецификата на предметната област (кой от моделите е най-подходящ за случая).

2.4.1 Йерархичен модел

В йерархичния модел данните са представени във вид на йерархични (дървовидни) структури. Всеки възел представлява съвкупност от атрибути, характеризиращи един или няколко класа от същности. Връзките са представени чрез естествените връзки в дървовидната структура.

Йерархичният модел е удобен с това, че осигурява естествен начин за моделиране на йерархични структури, които се срещат в заобикалящия ни свят. Той е достатъчно прост за използване, тъй като работещите в областта на обработката на данни са добре запознати с йерархични структури и във почти всички езици за програмиране се предлагат средства за работа с такива структури. Ето защо, на базата на този модел са разработени известни и продължително използвани СУБД.

Основно неудобство на модела е необходимостта от изкуствено представяне на връзка от тип “много - много”. Освен това, заради строгия йерархичен ред на връзките, се затрудняват основните операции с данните: добавяне, изтриване и актуализация.

2.4.2 Мрежов модел

При този модел класовете от същности се обединяват в "мрежа". Графично мрежовият модел се представя чрез ориентиран граф. Всеки възел на графа може да съдържа нула, един или няколко атрибута, характеризиращи обикновено един клас. Дъгите на графа отразяват връзките между класовете. Всяка дъга се именува, което

позволява една и съща двойка класове да участват в няколко взаимни връзки. За класовете, организирани в мрежов модел са характерни връзките от тип “много - много”. Йерархичният модел може да бъде разглеждан като частен случай на мрежовия модел.

На базата на този модел също са реализирани широкоразпространени СУБД.

2.4.3 Релационен модел

Данните при този модел се представят във вид на таблица, наричана в терминологията на релационния модел "отношение" или релация. Всеки стълб от

таблицата представлява един атрибут. Редовете на таблицата представляват наредени n-торки от стойности на съответните атрибути.

Поредицата от имена на стълбовете на таблицата съответства на списъка от атрибути, включени в разглежданата таблица и представя нейната структурата. Според терминологията на релационния модел е прието да се нарича схема на релацията.

Релационният модел се е появил последен в развитието на теоретичните основи на БД. Основава се на теорията на отношенията и средствата, които предлагат различните езици за програмиране за работа с множества.

Използваните напоследък СУБД са разработени на базата на този модел. В нашия курс ще отделим внимание основно на релационния модел.

2.5 Преобразуване на модела “Същност – връзка” в релационен модел.

Преобразуването на модела “Същност – връзка” в релационен модел се извършва по следните правила:

1. Всеки клас от стщности се представя в отделна таблица, която включва неговите атрибути. Идентификаторът на класа съответства на ключовия атрибут на таблицата.
2. Връзките от тип “много – много” се представят в отделна таблица, която включва идентификаторите на класовете, които връзката свързва, както и атрибутите на връзката, ако такива съществуват.
3. Връзка от тип “един – един” може да се представи в отделна таблица, която включва идентификаторите на класовете, учатстващи във връзката, но най-често се добавя към една от таблиците, съответстващи на класовете.
4. Връзка от тип “един – много” може да се представи в отделна таблица, включваща идентификаторите на класовете, които участват във връзката. Друг начин за представяне на такава връзка е обединяването и с една от таблиците, описващи някой от класовете, участващи във връзката.

3. Релационен модел. Релационна алгебра

3.1 Основни положения

Релационният модел се е появил най-късно в развитието на теоретичните основи на БД (през 1970г.). Главна причина за популярността му е фактът, че този модел предлага много мощен и, в същото време, ясен и лесен за възприемане начин за описване както на данните в БД, така и на операциите с тях. В основата на релационния модел стои математическото понятие n-членна релация. Всяка релация е множество от елементи, които се състоят от n компоненти, наречени n-торки (tuples).

Релационният модел предоставя възможността да се гледа на БД като съвкупност от релации, с които може да се оперира като се използват операциите от релационната алгебра. Релационният модел предоставя висока степен на независимост на данните, тъй като при него отсъства необходимостта да се познава вътрешното представяне. Съществено предимство на този модел пред другите модели на данните е еднообразното представяне на класовете от същности и връзките между тях.

Релационният модел се изгражда с три основни елемента: област (домен), атрибут и релация.

Нека D_1, D_2, ..., D_n са n, $n \geq 1$, множества, не непременно различни. Всяко от тези множества е именувано и може да съдържа елементи, които са допустими стойности за дадена величина. Такова множество се нарича **област** или **домен**. Областите могат да бъдат безкрайни или да съдържат краен брой елементи. Разглеждаме декартовото произведение на n-те множества D_1, D_2, ..., D_n:

$D_1 \times D_2 \times \ldots \times D_n = \{ <d_1, d_2, \ldots, d_n> | d_i \in D_i, i=1, 2, \ldots, n\}$.

Всяко подмножество на декартовото произведение се нарича **релация (relation)**. Ясно е, че в различни моменти от времето, съдържането на една релация може да бъде различно.

Отчитайки дефиницията на понятието релация, а именно, че всеки неин елемент е наредена n-торка $d_1, d_2, \ldots, d_n$, където d_i принадлежи на областта D_i, i=1, 2, ..., n, можем да разгледаме релацията и като таблица от елементи, в която редовете са n-торки, а стълбовете съдържат елементи само от една и съща област. При това представяне се изисква задължително спазване на местата на отделните стълбове от таблицата. Може обаче да се покаже, че ако стълбовете се именуват, то информационното съдържание на релацията се запазва и при размяна на местата

на стълбовете. Всеки именуван стълб на релацията се нарича **атрибут на релацията**.

3.2 Релационна схема. Схема на БД

Структурата на всяка релация се задава с нейната **релационна схема**. Релационната схема задава името на релацията и съответните и атрибути. Например, ако A_1, A_2, ..., A_n са атрибутите на релация с име R, нейната релационна схема ще се записва с израза $R(A_1, A_2, ..., A_n)$. Тъй като всеки атрибут може да получава стойност само от точно определена област, понякога, при необходимост, се използва и по-пълният запис $R(A_1:D_1, A_2:D_2, ..., A_n:D_n)$.

Нека $R(A_1, A_2, ..., A_n)$ е релационна схема, а r е релация с тази релационна схема. Този факт накратко се означава така: r:R.

Съвкупността от всички релационни схеми, с които се описват класовете от същности и връзките между тях, представляват **схемата на БД**. Ако R_1, R_2, ..., $R_к$ са релационни схеми, съставящи схемата на БД, това кратко се записва по следния начин: $(R_1, R_2, ..., R_к)$.

Всяка от тези релационни схеми има текуща стойност и това е съдържанието на съответната релация. Всяко множество от конкретни релации r_1, r_2, ..., $r_к$, за които r_i: R_i, i= 1, 2, ..., n, се нарича **текущо състояние на релационната БД** .

Сред атрибутите на дадена релация винаги има такива, по чиито стойности може еднозначно да се определят стойностите на останалите атрибути, т.е. еднозначно да се определи една n-торка от съвкупността n-торки. Те, поотделно или в различни комбинации, представляват **възможни ключови атрибути (ключове на релацията)**. Естествено е една релация да има повече от един ключ. Ако едно множество от атрибути A е ключ на една релация R, то всяко множество X от атрибути на R, за което $A \subset X$, също ще идентифицира еднозначно елементите на R. Затова към всеки ключ се предявява изискването за минималност, което може да се изрази по следния начин:

Ако K е ключ на релацията R, тогава:

1. За всеки два елемента $r_1 \in R$ и $r_2 \in R$ е в сила $r_1[K] \neq r_2[K]$.

2. Не съществува подмножество от атрибути на K, за което свойство 1. остава в сила.

Един от всички възможни ключове се избира за ключ на релацията и се нарича **първичен ключ (prymary key)**.

Външен ключ (foreign key) K* за една релация R е такъв атрибут, който не е ключ на R, но съществува релация Q от същата БД, за която K* е първичен ключ.

Първичните и външните ключове са основно средство за установяване на връзки между две релации.

3.3 Релационна алгебра

Релационната алгебра е набор от операции (релационни оператори), чрез които се описва начин за получаване на нови релации от други съществуващи релации.

3.3.1 Използвани понятия

Нека A_1, A_2, ..., A_n е списък от n атрибути, който ще означим кратко с A. Нека B е друг списък от m атрибута B_1, B_2, ..., B_m. Списъците от атрибути A и B са **сравними**, ако:

1. n=m

2. A_k и B_k са от един и същ тип за всяко k=1, 2, ..., n.

Нека p е n-членна релация с релационна схема R(A), а q е m-членна релация с релационна схема R(B), като a и b са елементи съответно от p и q, т. е. $a \in p$, $b \in q$ и

$a = <a_1, a_2, ..., a_n>$

$b = <b_1, b_2, ..., b_m>$.

Конкатенация на a с b е (n+m)-торка и е определена по следния начин:

$ab = <a_1, a_2, ..., a_n, b_1, b_2, ..., b_m>$.

3.3.2 Операции с релации

Обединение

Нека p и q са две n-членни релации с една и съща релационна схема $R(A_1, A_2, ..., A_n)$. Обединението на релациите p и q е трета релация r със същата релационна схема, съдържаща елементите на p и q, т. е.

$r = \{t \mid t \in p \text{ или } t \in q\}$.

Обикновено се означава с общоприетия в математиката знак за обединение на множества"$\cup$" :

$r = p \cup q$.

Пример 1.

Нека са дадени две релации r_1 и r_2 с еднакви релационни схеми R(Фамилия, Фак. Номер, Група).

r_1

Фамилия	Фак. номер	Група
Стоянова	026612	43
Илиев	036605	44
Дочева	036614	43

r_2

Фамилия	Фак. номер	Група
Илиев	036605	44
Славчева	036620	42
Христов	026617	44
Дочева	036614	43

Обединението на двете релации r_3, ще има следния вид:

$r_3 = r_1 \cup r_2$

Фамилия	Фак. номер	Група
Илиев	036605	44
Стоянова	026612	43
Славчева	036620	42
Христов	026617	44
Дочева	036614	43

Сечение

Нека p и q са две n-членни релации с една и съща релационна схема $R(A_1, A_2, ..., A_n)$. Сечението на релациите p и q е трета релация r със същата релационна схема, съдържаща тези елементи, които принадлежат и на p и на q, т. е.

$r = \{t \mid t \in p \text{ и } t \in q \}$.

Обикновено се означава с общоприетия в математиката знак за сечение на множества"$\cap$" :

$r = p \cap q$.

Пример 2.

Нека са дадени същите две релации r_1 и r_2 от **Пример 1**.

Сечението на двете релации r_3, ще има следния вид:

$r_3 = r_1 \cap r_2$

Фамилия	Фак. номер	Група
Илиев	036605	44
Дочева	036614	43

Разлика

Нека p и q са две n-членни релации с една и съща релационна схема R(A_1, A_2, ..., A_n). Разликата на релациите p и q е трета релация r със същата релационна схема, съдържаща елементите на p, непринадлежащи на q, т. е.

$r = \{t \mid t \in p \text{ и } t \notin q\}$.

Означава се със знак "-" :

$r = p - q$.

Пример 3.

Нека са дадени същите две релации ***r₁*** и ***r₂*** от **Пример 1**.

Разликата на двете релации ***r₃***, ще има следния вид:

r₃ = r₁ - r₂

Фамилия	Фак. номер	Група
Стоянова	026612	43

Декартово произведение

Нека p е релация с релационна схема P(A_1, A_2, ..., A_n), а q е релация с релационна схема Q(B_1, B_2, ..., B_m). Декартовото произведение на релациите p и q е трета (n+m)-членна релация r с релационна схема схема R(A_1, A_2, ..., A_n, B_1, B_2, ..., B_m), съдържаща всички възможни конкатенации на елементи от p и q, т. е.

$r = \{ab \mid a \in p \text{ и } b \in q\}$.

Означава се с "$\times$":

$r = p \times q$.

Пример 4.

Нека са дадени две релации ***r₁*** с релационна схема R(Фамилия, Фак. Номер, Група) (от Пример 1.) и ***r₂*** с релационна схема P(Дисциплина, Курс).

r₂

Дисциплина	Курс
БД	2
УП	1
ООП	1

Декартовото произведение на двете релации ***r₃***, ще има следния вид:

$r_3 = r_1 \times r_2$

Фамилия	Фак. номер	Група	Дисциплина	Курс
Стоянова	026612	43	БД	2
Илиев	036605	44	УП	1
Дочева	036614	43	ООП	1
Стоянова	026612	43	БД	2
Илиев	036605	44	УП	1
Дочева	036614	43	ООП	1
Стоянова	026612	43	БД	2
Илиев	036605	44	УП	1
Дочева	036614	43	ООП	1

Проекция

Нека p е релация с релационна схема $P(A_1, A_2, ..., A_n)$. Проекцията на релацията p по отношение на списъка от атрибути X ($X \in A$) е k-членна релация r с релационна схема $R(A_1, A_2, ..., A_k)$, която се получава от p чрез отстраняване на всички атрибути, които не са от X и в която всички повтарящи се редове са представени само един път.

Обикновено се означава с $\pi_X(p)$.

Пример 5.

Нека е дадена релация r_1 от Пример 1.

Проекцията r_3 ще има следния вид:

$r_3 = \pi_{\text{Група}}(r_1)$

Група
43
44

Селекция (рестрикция)

Нека p е релация с релационна схема $P(A_1, A_2, ..., A_n)$. Селекцията на релацията p по отношение на дадено условие F е друга релация r със същата релационна схема, всеки ред на която удовлетворява условието F, т. е.

$r = \{t \mid t \in p \text{ и } F(t) = \text{истина}\}$.

Очевидно е, че $r \subseteq p$. Селекцията се означава с $\sigma_F(p)$.

Пример 6.

Нека е дадена релация r_1 от Пример 1.

Селекцията r_3 ще има следния вид:

$r_3 = \sigma_{\text{Група=43}}(r_1)$

Фамилия	Фак. номер	Група
Стоянова	026612	43
Дочева	036614	43

Съединение

Нека p е релация с релационна схема $P(A_1, A_2, ..., A_n)$, а q е релация с релационна схема $Q(B_1, B_2, ..., B_m)$. Съединение или θ - съединение на релациите p и q по отношение на атрибутите A_i и B_j, $1 \le i \le n$, $1 \le j \le m$ е (n+m)-членна релация r с релационна схема $R(A_1, A_2, ..., A_n, B_1, B_2, ..., B_m)$. Релацията r е такова подмножество на декартовото произведение $p \times q$, при което A_i - тия атрибут на r е в θ отношение с B_j - тия атрибут. С θ е означено кое да е от отношенията за сравнение.

За означаване на съединението се използва "$\triangleright\triangleleft$":

$$r = p \underset{A_i\ \theta\ B_j}{\triangleright\triangleleft} q = \sigma_{A_i\ \theta\ B_j}(p \times q).$$

Пример 7.

Нека са дадени две релации r_1 и r_2 от Пример 4.

Съединението на двете релации r_3, ще има следния вид:

$r_3 = r_1 \triangleright\triangleleft r_2$

Фак.номер>'030000' и Курс=1

Фамилия	Фак. номер	Група	Дисциплина	Курс
Илиев	036605	44	УП	1
Дочева	036614	43	ООП	1
Илиев	036605	44	УП	1
Дочева	036614	43	ООП	1
Илиев	036605	44	УП	1
Дочева	036614	43	ООП	1

Естествено съединение

Нека p и q са две релации, които имат общ атрибут C в релационните си схеми, съответно $P(A_1, A_2, ..., A_n, C)$ и $Q(C, B_1, B_2, ..., B_m)$. Естественото съединение на релациите p и q е трета (n+m+1)-членна релация r с релационна схема $R(A_1, A_2, ..., A_n, C, B_1, B_2, ..., B_m)$, която се определя по формулата

$$r = \pi_Z(\sigma_F(p \times q))$$

където Z е списъкът от атрибутите A_1, A_2, ..., A_n, C, B_1, B_2, ..., B_m, а F е условието за съединение, което има вид p.C = q.C. Условието p.C = q.C означава, че съединението ще се извърши само между онези редове от p и q, за които атрибутът C има една и съща стойност.

Операцията естествено съединение може да се прилага не само над единствен атрибут, а и над списък от атрибути.

Пример 8.

Нека са дадени релациите ***r1*** (от Пример 1.) и ***r2*** с релационна схема P(Група, Дисциплина).

r2

Група	Дисциплина
43	СДП
42	БД
43	УП
44	ООП

Естественото съединение на двете релации ***r3***, ще има следния вид:

r3 = r1 ▷◁ r2

Фамилия	Фак. номер	Група	Дисциплина
Стоянова	026612	43	СДП
Стоянова	026612	43	УП
Дочева	036614	43	СДП
Дочева	036614	43	УП

4. Релационни езици. Езици за работа с данните

4.1 Процедурен и непроцедурен подход

Операциите от релационната алгебра са средството, с помощта на което се описват действията, които е необходимо да се изпълнят върху релациите, влизащи в състава на конкретна БД, за да се получи определен резултат. Релационната алгебра може да се раглежда като релационен език от високо ниво. Все пак е необходимо да се отбележи, че твърде малък брой СУБД предлагат езици, базирани изцяло на релационната алгебра. Дефинирайки заявка като израз от релационната алгебра потребителят указва **как** да се изпълни заявката, т.е. кои операции и в какъв ред да се изпълнят. Този подход се нарича процедурен, тъй като потребителят е длъжен да дефинира точна процедура, изпълнението на която ще доведе до желания резултат. Оказва се, че предпочитанията на потребителите са към използване на декларативни езици от високо ниво, чрез които се дефинира **какво** да се получи, а какви точно операции да се използват и в каква последователност да се изпълнят се решава от СУБД. В тях е заложен непроцедурен подход, тъй като на потребителя се предоставят средства за описване на желания резултат.

На практика рядко се срещат езици, които са изцяло процедурни или изцяло непроцедурни. Най-често се използват смесени езици, които предлагат на потребителя както средства за дефиниране директно на последователност от релационни оператори, така и средства за описание на свойствата на резултата, които трябва да се получи след изпълнение на заявката.

Ще разгледаме два широко използвани езика за зявки.

4.2 SQL

SQL (Structured Query Language) се е наложил като стандарт за работа с релационни БД. Често различните СУБД използват диалекти на езика.

За разлика от релационната алгебра, където могат да се дефинират само заявки за справки, SQL е пълен език. Освен операторите от за дефиниране на заявки, той обхваща и оператори, съставящи ЕОД (Език за описание на данните), както и оператори за управление и администриране на БД.

Операторите на езика основно се групират в следните пет групи:

- оператори за описание на данните: CREATE TABLE, DROP TABLE, ALTER TABLE, CREATE VIEW, DROP VIEW, ALTER VIEW, CREATE INDEX, DROP INDEX;
- оператори за манипулиране с данните: INSERT, UPDATE, DELETE;
- оператор за заявки: SELECT;
- оператори за управление на транзакции: COMMIT, ROLLBACK, SAVEPOINT;
- оператори за администриране на данните: ALTER DATABASAE, ALTER DBAREA, ALTER PASSWORD, CREATE DATABASE, CREATE DBAREA, DROP DATABASE, DROP DBAREA, GRANT, REVOKE.

Комерсиалните СУБД предлагат и допълнителни оператори.

4.2.1 Оператор SELECT

Заявките в SQL се дефинират с помощта на единствен оператор – SELECT. Чрез него е възможно задаването на действия, които се реализират с помощта на всички операции от релационната алгебра. Също така е възможно и дефинирането на допълнителни заявки, които изискват използването на т.н. обобщаващи функции. Макар че за моделиране на заявките се използва един единствен оператор, това не означава, че това моделирането на заявките е прост процес. Една и съща заявка може да се опише по много и различни начини, което се отразява на времето за тяхното изпълнение, което е твърде съществено, особено за големи БД. Ето защо създаването на заявки изисква определена теоретична подготовка, както и практика.

Синтаксисът на оператора SELECT има следния вид:

SELECT [All | Distinct] (<Списък от имена на полета> | *)
FROM <Списък от имена на таблици>
[WHERE <Критерий за избор>]
[GROUP BY <Списък от имена на полета>]
[HAVING <Условие за група>]
[ORDER BY <Списък от имена на полета, по които се извършва подреждане];

Тук ключовата дума All означава, че ще се изведе информация от всички редове, които удовлетворяват заявката, ключовата дума Distinct означава, че е необходимо да се изведат само различни редове в получения резултат.

Символът * се използва за кратко означаване на всички колони на входните таблици.

Разделът FROM се използва за указване на имената на входните таблици, т.е. таблиците, които включват участващите в заявката атрибути, както и тези, които се използват за свързване на други таблици, когато това е необходимо.

Разделът WHERE служи за задаване на критерий за избор, чрез който ще се отделят само редовете, удовлетворяващи определени условия. За дефиниране на условия се използват следните предикати:

- сравнения, които се означават чрез знаците =, <>, >, <, >=, <= и имат традиционния смисъл;
- Between A and B – използва се за означаване на стойности, които са в диапазона между A и B. За указване на стойности извън даден интервал се използва противоположния предикат Not Between A and B;
- IN – използва се за означаване на принадлежност към множество. Съществува и противоположен предикат Not IN. Тези предикати най-често се използват за принадлежност към подзаявка;
- Like – използва се за задаване на сравнение по образец. Not Like има противоположен смисъл;
- Is Null – използва се за сравнение с неопределена стойност. Неопределена стойност според терминологията на релационните БД е тази, която не е известна в определен момент.
- Exist – използва се за указване на съществуване или несъществуване (Not Exist). Прилагат се най-често при работа с подзаявки.

Важно е да се отбележи, че, както се вижда от дефиницията на оператора SELECT, само ключовите думи SELECT и FROM са задължителни. Останалите раздели се използват когато е необходимо.

Примери:

Нека е дадена БД със следната схема:

Купувач (К#, Име, Адрес)

Поръчка (П#, Дата)

Стока (С#, Название, Ед. Цена)

Поръчва (К#, П#)

Включва (П#, С#, Поръчано количество)

SELECT *К#* FROM *Купувач*;

Ще се изведат номерата на всички купувачи, представени в таблицата ***Купувач.***

SELECT *К#* FROM *Купувач, Поръчка* WHERE *Купувач.К# = Поръчва. П#*;

Ще се изведат номерата на всички купувачи, представени в таблицата ***Купувач***, които се срещат и в таблицата ***Поръчва***, т.е. номерата на тези купувачи, които са направили поне една поръчка. Тази заявка може да се формулира и по следния начин:

SELECT *К#* FROM *Купувач* WHERE *К#* IN (SELECT *К#* FROM *Поръчва)*;

В този случай се използва подзаявка, която определя множество от номерата на купувачите, направили поне една поръчка. След това, от таблицата ***Купувач*** се отделят само тези номера на купувачи, които присъстват в дефинираното чрез подзаявката множество.

SELECT *Име* FROM *Купувач* WHERE *К# > 1000*;

Ще се изведат имената на всички купувачи, представени в таблицата ***Купувач***, чиито номера са по-големи от 1000.

SELECT *К#, С#* FROM *Купувач, Стока*;

Ще се изведат всички двойки К# и С#, които се срещат в декартовото произведение на таблиците Купувач и Стока. Тук, между таблиците Купувач и Стока ще се изпълни операция декартово произведение, тъй като те нямат общ атрибут.

SELECT *К#, П#* FROM *Купувач, Поръчва*;

Ще се изведат всички двойки ***К#*** и ***П#,*** които се срещат в декартовото произведение на таблиците ***Купувач*** и ***Поръчва***. Тук, между таблиците ***Купувач*** и ***Поръчва*** ще се изпълни операция декартово произведение, тъй като за тях, макар че те имат общ атрибут, не е зададено условие за съединяване.

4.2.2 Използване на обобщаващи функции

SQL предлага на потребителя вградени функции, с помощта на които се изчисляват обобщаващи групови стойности. Използването на тези функции предполага предварително изпълнение на операция групиране. При нея множеството от редовете на обработваната таблица се разделя на групи с еднакви стойности на атрибутите, по които се извършва групирането.

Пример:

За да определим броя на поръчките, направени на всяка дата, ще използваме таблицата Поръчка, към която ще приложим групиране по дата:

SELECT *Count (П#)* FROM *Поръчка GROUP BY Дата*;

Докато чрез оператора

SELECT *Count (П#)* FROM *Поръчка*;

ще се изведе общият брой на поръчките, за които в БД има информация.

А с помощта на оператора

SELECT *Count (П#)* FROM *Поръчка* WHERE *Дата > '20.10.2002'*;

ще се изведе общият брой на поръчките, които са направени след 20 октомври 2002 година.

Чрез използване на NAVING е възможно доуточняване на извежданите стойности на обобщаващите функции.

Пример:

Ако е необходимо да определим номерата на купувачите, които са направили повече от 5 поръчки, ще използваме следния оператор:

SELECT *К#* FROM *Поръчва* GROUP BY *К#* HAVING *Count (П#) > 5*;

4.2.3 Оператори за манипулиране с данните

Операторът за въвеждане на данни има следния синтаксис:

INSERT INTO име на таблица [(<списък от имена на полета>)] VALUES (<списък от стойности>);

Операторът се използва за въвеждане на един ред от указаната таблица. Списъкът от имена на полета задава полетата, на които трябва да се присвоят зададените стойности.

Пример:

INSERT INTO ***Купувач*** (***К#, Име)*** **VALUES** ('1010', 'Славеев');

В резултат от изпълнението на този оператор в таблицата ***Купувач*** ще се появи ред, описващ купувач с име Славеев и номер 1010. Полето ***Адрес*** ще остане с неопределена стойност, тъй като такава не е зададена в оператора.

Операторът за отстраняване на данни има следния синтаксис:

DELETE FROM име на таблица [WHERE <Критерий за избор>];

Критерият за избор тук има същия смисъл както в оператор SELECT и определя редовете на таблицата, които трябва да се отстранят. Ако той отсъства, тогава се отстраняват всички редове на таблицата, но това не означава, че самата таблица се отстранява от БД. Тя остава, но е празна.

Пример:

DELETE FROM ***Купувач*** **WHERE** ***К#***< 1010;

В резултат от изпълнението на този оператор от таблицата ***Купувач*** ще се отстранят всички редове, в които номерата на купувачите са по-малки от 1010.

Операторът за обновяване на данни има следния вид:

UPDATE име на таблица SET име на поле = нова стойност

[WHERE <Критерий за избор>];

Операторът се използва за задаване на нова стойност на определено поле от зададена таблица. Новата стойност ще се установи във всички редове на таблицата, които удовлетворяват критерия за избор.

Пример:

UPDATE ***Включва*** **SET** ***Поръчано количество*** = 2 **WHERE** ***С#*** =1010;

В резултат от изпълнението на този оператор от таблицата ***Включва*** за всяка поръчка, в която присъства стока с номер 1010, ***Поръчано количество*** ще приеме стойност 2.

4.3 QBE

QBE (Query By Example) е език, който се използва най-често в графична среда. Той предлага на потребителя графичен инструментариум за моделиране на заявка чрез попълване на шаблони, които съответстват на таблиците, включени в БД.

За разгледаната по-горе БД, шаблоните ще изглеждат така:

Купувач	***К#***	***Име***	***Адрес***

Поръчка	***П#***	***Дата***

Стока	***С#***	***Название***	***Ед. цена***

Поръчва	***К#***	***П#***

Включва	***П#***	***С#***	***Поръчано к-во***

Потребителят не е задължен да запомня нито имената на таблиците, нито имената на полетата, те се извеждат върху дисплея, както е показано по-горе. Също така от него не се изисква да познава синтаксиса на определени оператори. Той може да конструира желаната заявка като пример, на базата на зададените шаблони.

4.3.1 Извличане на данни

Заявките за извличане на данни се дефинират чрез попълване на определени стойности в някои от колоните на шаблоните, както и на определени команди, чрез които се задават действия, подлежащи на изпълнение.

Ще се спрем най-напред на извличане на данни. За целта се използва команда **P.**, която се поставя под името на колоната, стойностите на която

трябва да се изведат. Ако е необходимо извеждането на всички колони от дадена таблица, командата може да се постави и под името на таблицата.

Примери:

Купувач	*К#*	*Име*	*Адрес*
	Р.		

Ще се изведат номерата на всички купувачи, представени в таблицата ***Купувач.***

Купувач	*К#*	*Име*	*Адрес*
	>1000	**Р.**	

Ще се изведат имената на всички купувачи, представени в таблицата ***Купувач***, чиито номера са по-големи от 1000.

Когато е необходимо в заявката да се използват повече от една таблици, се задава условие за свързването им. Това става чрез поставяне на една и съща променлива в полетата на всяка от таблиците, чрез които става свързването.

За извеждането на имената на купувачите, които са направили поне една поръчка е необходимо използването на таблиците Купувач и Поръчва. В някои от колоните на тези таблици се поставят стгойности или команди, а в колоните, по които се извършва свърването, се поставя име на променлива:

Купувач	*К#*	*Име*	*Адрес*
	Х	**Р.**	

Поръчва	*К#*	*П#*
	Х	

Променливата Х в двата шаблона показва, че необходимо да се извърши свързване на двете таблици по този атрибут, след което да се изведат стойностите на полето ***Име***.

4.3.2 Манипулиране с данните

Въвеждането, изтриването и обновяването на данни се задава аналогично на извличането, като за всяко действие се използва съответна команда.

Купувач	*К#*	*Име*	*Адрес*
I.	**1010**	**Славеев**	

След изпълнението на тази заявка в таблицата ***Купувач*** ще се появи ред, описващ купувач с име Славеев и номер 1010. Полето ***Адрес*** ще остане с неопределена стойност.

Купувач	*К#*	*Име*	*Адрес*
D.	**<1010**		

От таблицата ***Купувач*** ще се отстранят всички редове, в които номерата на купувачите са по-малки от 1010.

Включва	*П#*	*С#*	*Поръчано к-во*
		1010	**U.2**

В резултат от изпълнението на този оператор от таблицата ***Включва*** за всяка поръчка, в която присъства стока с номер 1010, ***Поръчано количество*** ще приеме стойност 2.

5. Обработка на заявките. Оптимизация

5.1 Основни положения

При изпълнение на заявките винаги се извършва преобразуване на израза зададен от потребителя в поредица от операции от релационната алгебра. Обикновено заявките са записани на някакъв език за работа с данните, използван от конкретна СУБД. Всяка заявка може да се изпълни по няколко различни начина, т. е. може да бъде представена като различни поредици от операции. Естествено е да се търси оптималната от тези последователности. Потребителят е в състояние да формулира своите заявки, така че те да се изпълняват по възможно най-ефективния начин, но за предпочитане е ако тази функция може да бъде възложена на СУБД. Системата ще преобразува въведените от потребителя заявки в еквивалентни, чието изпълнение е по-ефективно. Това преобразуване се нарича оптимизация на заявките. Оптимизацията на заявките е важна, тъй като разликата във времето за изпълнение на една неоптимизирана и еквивалентната и оптимизирана заявка може да бъде твърде голяма.

За оптимизиране на дадена заявка, най-напред тя се преобразува във вътрешно представяне (обикновено израз от операции на релационната алгебра), при което най-напред се извършва синтактичен контрол на заявката. След това се търси еквивалентен или еквивалентни изрази. За всеки израз се пресмята "цена" на изпълнението и, въз основа на тези пресмятания, се избира оптималния начин за изпълнение на заявката.

5.2 Еквивалентност на изразите

Основен етап от избирането на стратегия за изпълнение на заявката е намирането на еквивалентна поредица от операции на релационната алгебра. Този етап може да се разгледа като изпълнение на няколко отделни стъпки. На всяка стъпка се изследва наличието на отделни операции или комбинации от операции в израза и те се заместват с други операции (комбинации от операции), при пресмятането на които се получава същия резултат.

Съществуват редица правила, които се прилагат с цел да се получат еквивалентни изрази. Най-често се прилагат следните правила:

1. Комутативност на декартовото произведение:

$$r_1 \times r_2 \equiv r_2 \times r_1$$

2. Комутативност на съединението:

$$r_1 \rhd\lhd r_2 \equiv r_2 \rhd\lhd r_1$$

$$r_1 \overset{F}{\rhd\lhd} r_2 \equiv r_2 \overset{F}{\rhd\lhd} r_1$$

3. Асоциативност на декартовото произведение:

$$(r_1 \times r_2) \times r_3 \equiv r_1 \times (r_2 \times r_3)$$

4. Асоциативност на съединението:

$$(r_1 \rhd\lhd r_2) \rhd\lhd r_3 \equiv r_1 \rhd\lhd (r_2 \rhd\lhd r_3)$$

$$(r_1 \overset{F_1}{\rhd\lhd} r_2) \overset{F_2}{\rhd\lhd} r_3 \equiv r_1 \overset{F_1}{\rhd\lhd} (r_2 \overset{F_2}{\rhd\lhd} r_3)$$

5. Комбинация от проекции:

$$\pi_A(\pi_B(r)) \equiv \pi_B(r)$$

където множеството от атрибути A е подмножество на множеството от атрибути B.

6. Комбинация от селекции:

$$\sigma_{F1}(\sigma_{F2}(r)) \equiv \sigma_{F1 \wedge F2}(r)$$

7. Разместване на проекция и селекция:

$$\sigma_F(\pi_A(r)) \equiv \pi_A(\sigma_F(r))$$

8. Разместване на селекция и декартово произведение:

а) $\sigma_F(r_1 \times r_2) \equiv (\sigma_F(r_1)) \times r_2$

ако F съдържа атрибути само от r_1.

б) $\sigma_F(r_1 \times r_2) \equiv r_1 \times (\sigma_F(r_2))$

ако F съдържа атрибути само от r_2.

в) $\sigma_F (r_1 \times r_2) \equiv \sigma_{F1} (r_1) \times \sigma_{F2} (r_2)$

ако F може да се представи като F1 $\wedge$ F2 , където F1 съдържа атрибути само от r_1, а F2 съдържа атрибути само от r_2.

г) $\sigma_F (r_1 \times r_2) \equiv \sigma_{F2} (\sigma_{F1} (r_1)) \times r_2)$

ако F1 съдържа атрибути само от r_1, а F2 съдържа атрибути и от r_1 и от r_2.

9. Разместване на селекция и обединение:

$$\sigma_F (r_1 \cup r_2) \equiv \sigma_F (r_1) \cup \sigma_F (r_2)$$

където r_1 и r_2 имат еднакви релационни схеми.

10. Разместване на селекция и разлика:

$$\sigma_F (r_1 - r_2) \equiv \sigma_F (r_1) - \sigma_F (r_2)$$

където r_1 и r_2 имат еднакви релационни схеми.

11. Разместване на проекция и декартово произведение:

$$\pi_A (r_1 \times r_2) \equiv \pi_B (r_1) \times \pi_C (r_2)$$

където атрибутите B и C влизат в състава на A, като атрибутът B е от релационната схема на r_1, а атрибутът C е от релационната схема на r_2.

12. Разместване на проекция и обединение:

$$\pi_A (r_1 \cup r_2) \equiv \pi_B (r_1) \cup \pi_C (r_2)$$

където r_1 и r_2 имат еднакви релационни схеми и атрибутите B и C влизат в състава на A, като атрибутът B е от релационната схема на r_1, а атрибутът C е от релационната схема на r_2.

5.3 Оценяване на заявките

Стратегията, която се избира за изпълнение на дадена заявка, зависи от размерите на релациите, участващи в заявката и от стойностите на отделните атрибути. За да може да бъде избрана "добра" стратегия, СУБД съхранява статистическа информация за всяка релация в БД. Тази информация включва:

n_r - брой на n-торките (редовете) на релацията;

s_r - размер на всяка n-торка (обикновено в байтове);

V(A,r) - брой на различните стойности за атрибута A, които се съхраняват в БД.

Тези величини се използват за изчисляване на размерите на временните отношения, които се получават в резултат на извършване на всяка операция.

За определяне на размера на декартовото произведение са достатъчни първите две величини. Декартовото произведение на две релации p и q ($p \times q$) съдържа n_p . n_q реда. Всеки ред на $p \times q$ съдържа $s_p + s_q$ байта.

Размерът на временна релация, получена в резултат на операцията селекция, зависи и от третата величина, тъй като броят на редовете на такава релация се определя в зависимост от броя (честотата) на срещане на определена стойност на даден атрибут в изходните релации. Необходимо е да се изрази по някакъв начин честотата на срещане на всяка стойност на атрибутите, определящи селекцията. Ясно е, че присъствието на всяка стойност не е равновероятно, но обикновено, за да се опростят пресмятанията, се приема за такова. При това допускане селекцията $\sigma_{A=a}(r)$ ще има

$$\frac{n_r}{V(A,r)}$$

реда.

Оценката на размера на резултата от естественото съединение се извършва по-сложно от оценката за декартовото произведение и селекцията. Ще разгледаме две релации с релационни схеми p(P) и q(Q). Ако двете релации нямат общи атрибути, то естественото съединение съвпада с декартовото произведение и в този случай може да се използва методиката за оценяване на декартовото произведение. Ако $P \cap Q$ е ключ на едната релация, например на p, тогава редовете на q ще се съединяват точно с един ред от p, т.е. броят на редовете на $p \triangleright\triangleleft q$ няма да е по-голям от броя на редовете на q. Най-сложен е случаят, при който $P \cap Q$ не е ключ нито на p, нито на q. Ако $P \cap Q = A$, тогава за всеки ред от p в $p \triangleright\triangleleft q$ ще се получат

$$\frac{n_q}{V(A,q)}$$

реда в $p \triangleright\triangleleft q$, тъй като в q има толкова реда с различни стойности за A. След като в p има n_p реда, то тогава $p \triangleright\triangleleft q$ ще има

$$\frac{n_p \cdot n_q}{V(A,q)}$$

реда.

Ако направим същите разсъждения при разменени роли на p и q, ще получим оценка за брой на редовете на $p \triangleright\triangleleft q$

$$\frac{n_p \cdot n_q}{V(A,p)}$$

Тези две оценки се различават, тъй като обикновено $V(A,p) \neq V(A,q)$. Вижда се, че оценката на броя на редовете е приблизителна, въпреки това изчисляването и е полезно.

За да се извършва актуално оценяване на всяка заявка, заедно с промените в БД, е необходимо да се обновява и статистическата информация, използвана за изчисляване на оценките. Такова обновяване не се извършва след всяка модификация на БД, а периодично, през определени интервали от време. Ето защо оценките не са много точни, но могат да се приемат за удовлетворителни.

Освен разгледаната статистическа информация, СУБД използват за оценяване на заявките и информация за физическото разположение на данните в БД.

Съществуват различни методи за оценяване на заявките с отчитане на физическата организация на БД. Това позволява да се правят по-обхватни оценки.

За да се опрости изчисляването на оценките за изпълнението на заявките, обикновено една заявка се разбива на подзаявки, всяка от които поотделно се оценява. Това не само опростява процеса на оценяване, но и позволява да се открият случаите, при които една подзаявка се среща (трябва да се изпълни) няколко пъти. Разпознаването на общи подзаявки позволява съществено да се намали времето за оценяване и изпълнение.

6. Анализ на релационни схеми

6.1 Основни положения

Основен проблем при изграждане на модел на дадена предметна област е определянето на същностите, които съставляват предметната област и на свойствата, които ги характеризират. Сложността на проблема е следствие от неоднозначността на неговото решение. Естествени, в този случай, са следните въпроси: кога една релационна схема е добра; кога две схеми са еквивалентни; коя от две схеми е по-добра. Отговори на тези въпроси могат да се получат в резултат от анализиране на релационни схеми.

При анализирането и оценяването на релационни схеми се изследва наличието на различни недостатъци в тези схеми. В литературата тези недостатъци се определят като:

- аномалии при излишество:
- аномалии при обновяване;
- аномалии при добавяне;
- аномалии при отстраняване.

Ще разгледаме примери за такива аномалии. Нека е дадена релационната схема:

Стоки (Код_на_купувача, Име_на_купувача, Код_на _стоката, Наименование_на_стоката, Количество),

в която всеки купувач е представен чрез ***Код*** и ***Име***, всяка стока - чрез ***Код*** и ***Наименование***. Количество определя количеството от дадена стока, закупено от даден купувач.

Направени са следните допускания:

- Всеки купувач еднозначно се определя от своя код;
- Всяка стока еднозначно се определя от своя код;
- Една и съща стока може да бъде закупена от различни купувачи.

Като следствие от тези допускания могат да се направят следните изводи:

1. За всяка стока, закупена от даде купувач, се повтарят кодът и името на купувача.

2. Ако се промени кодът на купувача, то трябва да се променят толкова реда, колкото стоки е закупил той.

3. Информация за даден купувач (Код и Име) не може да бъде въведена преди той да е закупил някаква стока.

4. Ако трябва да се отстрани информация за всички стоки, които е закупил даден купувач, то ще се отстрани и информацията за самия него.

Посочените недостатъци съответстват на аномалиите, споменати по-горе.

Възможно е схемата да се преобразува така, че да се отстранят отчасти или напълно някои от споменатите аномалии. Методите за "подобряване" на релационните схеми се основават на изследване на различните зависимости, съществуващи явно или неявно между атрибутите на една релационна схема. Разглеждат се два вида зависимости - функционални и многозначни.

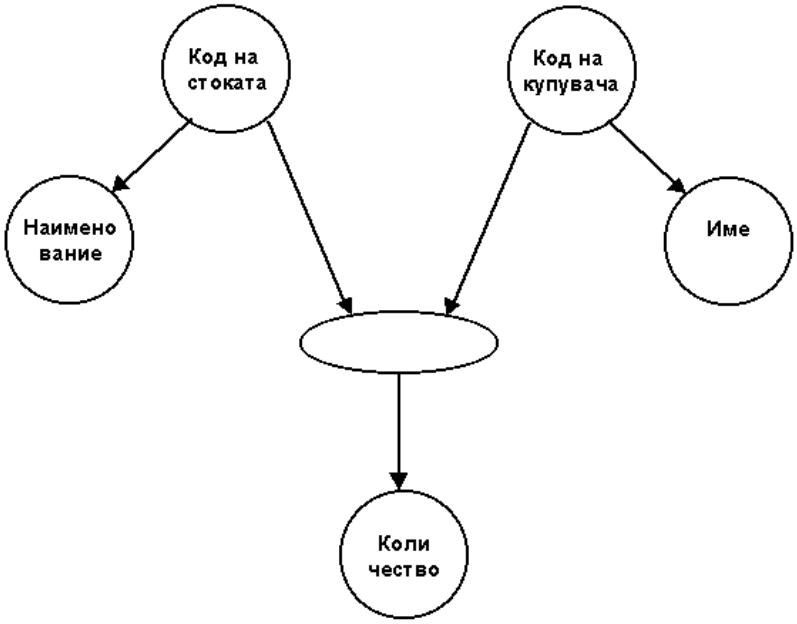
6.2 Функционални зависимости

Нека R(A) е релационна схема и $X \subseteq A$ и $Y \subseteq A$. Казва се, че атрибутът X функционално определя атрибута Y, ако за всяка релация r с релационна схема R(A) при наличието на кои да е два реда t_1, $t_2 \in r$, за които $t_1.X = t_2.X$, следва, че $t_1.Y = t_2.Y$, т.е. при равенство на редове за атрибута X, следва равенство на редове за атрибута Y. Когато атрибутът X функционално определя атрибута Y, казва се още, че Y функционално зависи от X, което се означава така: $\mathbf{X \rightarrow Y}$.

Формално една функционална зависимост (**F-зависимост**) може да се изрази чрез операции от релационната алгебра така: за релация r с релационна схема R(A), ако $|\pi_Y(\sigma_{X=x}(r))| \leq 1$ за всяко x, $x \in X$, то релацията r удовлетворява функционалната зависимост $X \rightarrow Y$.

Всяка релация удовлетворява множество от функционални зависимости, което е прието да се означава чрез **F**.

Удобно е множеството от F-зависимости да се представи нагледно чрез граф G=(A, F), където A е множеството от атрибути, а F - множеството от функционални зависимости за дадена релация. На всеки атрибут е съпоставен връх от графа. Ако X е прост атрибут, тогава на функционалната зависимост $X \rightarrow Y$ се съпоставя дъга. Ако X е съставен атрибут, се въвеждат върхове от втори вид, съответстващи на съставните атрибути (изобразени чрез овал).



6.3 Правила (аксиоми) за извод

За всяка релация r в даден момент съществува множество от F-зависимости, които релацията удовлетворява. Обикновено едно състояние на релацията може да удовлетворява едно множество от F-зависимости, а друго състояние да не удовлетворява същото множество от F-зависимости. Необходимо е да се намери такова множество от F-зависимости, които се удовлетворяват от всички състояния на релацията.

У. Армстронг показва, че е възможно, при наличието на дадено множество от F-зависимости, да се получат други F-зависимости, които дадената релация също удовлетворява. При това могат да се получат всички F-зависимости.

Ще формулираме аксиомите за извод на F-зависимости, като отчитаме, че: R(A) е релационна схема, F е множество от F-зависимости, X, Y, Z, W са атрибути (прости или съставни) и са подмножества на A.

F1. Рефлексивност

$X \rightarrow X$.

Релацията $\pi_X(\sigma_{X=X}(r))$ винаги има не повече от един ред, следователно r удовлетворява $X \rightarrow X$.

F2. Попълнение

Ако $X \rightarrow Y$, то $XZ \rightarrow Y$.

Тази аксиома е свързана с разширяване на лявата част на F-зависимостта. Ако r удовлетворява $X \rightarrow Y$, то $\pi_Y(\sigma_{X=x}(r))$ има не повече от един ред за всяка стойност на X. Ако Z е подмножество на A, то $\pi_Y(\sigma_{XZ=xZ}(r)) \subseteq \pi_Y(\sigma_{X=x}(r))$. Следователно $\pi_Y(\sigma_{XZ=xZ}(r))$ има не повече от един ред или r удовлетворява $XZ \rightarrow Y$.

F3. Обединение

Ако $X \rightarrow Y$ и $X \rightarrow Z$, то $X \rightarrow YZ$.

Тази аксиома позволява да се обединят две F-зависимости с еднакви леви части. Ако релацията r удовлетворява $X \rightarrow Y$ и $X \rightarrow Z$, то двете проекции $\pi_Y(\sigma_{X=x}(r))$ и $\pi_Z(\sigma_{X=x}(r))$ имат не повече от един ред за всяка стойност на X. Ако релацията $\pi_{YZ}(\sigma_{X=x}(r))$ има повече от един ред, то поне една от релациите $\pi_Y(\sigma_{X=x}(r))$ и $\pi_Z(\sigma_{X=x}(r))$ би имала повече от един ред. От това следва, че r удовлетворява F-зависимостта $X \rightarrow YZ$

F4. Проективност

Ако $X \rightarrow YZ$, то $X \rightarrow Y$.

Тази аксиома в известна степен е обратна на F3. Ако r удовлетворява $X \rightarrow YZ$, то релацията $\pi_{YZ}(\sigma_{X=x}(r))$ има не повече от един ред за всяка стойност на X.Тъй като $\pi_Y(\pi_{YZ}(\sigma_{X=x}(r))) = \pi_Y(\sigma_{X=x}(r))$, то $\pi_Y(\sigma_{X=x}(r))$ също има не повече от един ред. Следователно, r удовлетворява $X \rightarrow Y$.

F5. Транзитивност

Ако $X \rightarrow Y$ и $Y \rightarrow Z$, то $X \rightarrow Z$.

Тази и следващата аксиоми са най-мощните аксиоми за извод. Нека r удовлетворява $X \rightarrow Y$ и $Y \rightarrow Z$ и нека в релацията r съществуват редове, за които $t_1.X=t_2.X$. От $X \rightarrow Y$ следва, че $t_1.Y=t_2.Y$, а от $Y \rightarrow Z$ следва, че $t_1.Z=t_2.Z$, т.е. от равенството $t_1.X=t_2.X$ следва равенството $t_1.Z=t_2.Z$ или r удовлетворява $X \rightarrow Z$.

F6. Псевдотранзитивност

Ако X→Y и YZ→W, то XZ→W.

Нека r удовлетворява F-зависимостите X→Y и YZ→W и нека в релацията r съществуват редове, за които $t_1.X=t_2.X$. По условие ако $t_1.X=t_2.X$, то $t_1.Y=t_2.Y$ и аналогично, ако $t_1.YZ=t_2.YZ$, то $t_1.W=t_2.W$. След като $t_1.XZ=t_2.XZ$, то $t_1.X=t_2.X$ и $t_1.Z=t_2.Z$. От $t_1.X=t_2.X$ следва и $t_1.Y=t_2.Y$, т.е. $t_1.YZ=t_2.YZ$ също е вярно. От $t_1.YZ=t_2.YZ$ следва $t_1.W=t_2.W$ и следователно r удовлетворява XZ→W.

Някои аксиоми могат да бъдат изведени от други. Например, F5 е частен случай на F6 при Z= ∅. Аксиомата F6 следва от F1, F2, F3 и F5: ako X→Y и YZ→W, то от F1 следва Z→Z. Съгласно F2 имаме XZ→Y и XZ→Z. Прилагайки F3 ще получим XZ→YZ. И накрая, използвайки F5 ще получим XZ→W.

Аксиомите F1, F2 и F6 позволяват от тях да бъдат изведени останалите аксиоми. Освен това, аксиомите F1, F2 и F6 са независими - нито една от тях не може да бъде изведена от останалите. (Тези аксиоми в литературата са известни като аксиоми на Армстронг, макар че се различават от дефинираните първоначално от него.)

Всяка F-зависимост, която се удовлетворява от r, или принадлежи на предварително зададено множество от функционални зависимости, или може да бъде изведена чрез последователно прилагане на аксиомите за извод. Нека F е множество от функционални зависимости над r. Последователността P от F-зависимости над r е последователност за извод, ако всяка F-зависимост в P е:

1. или член на F

2. или е следствие на предходни F-зависимости от P след прилагане на една от аксиомите за извод F1-F6. P е последователност за извод на X→Y, ако X→Y е една от F-зависимостите в P.

За извод на F-зависимости се използва и друг набор от аксиоми, който не съвпада с набора F1-F6. Наричат се B-аксиоми:

B1. Рефлексивност (Reflexivity)

X→X.

B2. Натрупване (Accumulation)

Ако X→YZ и Z→CW, то X→YZC.

B3. Проективност (Projectivity)

Ако X→YZ, то X→Y.

От B-аксиомите може да се получат т.н. RAP-последователности за извод при следните условия:

1. Първа F-зависимост е винаги X→X.

2. Последна F-зависимост е X→Y.

3. Всяка F-зависимост, различна от първата и последната, или е вече в F, или има вида X→Z и е получена чрез B2.

6.4 Обвивки и покрития

6.4.1 Обвивки

Прилагайки аксиомите за извод към множеството от функционални зависимости за дадена релация r, можем да допълним това множество с други зависимости, които r също удовлетворява. **Обвивка на F**, означавана с **F^+**, е множеството от функционални зависимости с най-малко елементи и такова, че, чрез прилагане на аксиомите за извод, към него не могат да се добавят нови елементи. F^+ е крайно множество и може да бъде получено от F чрез прилагане на аксиомите F1, F2 и F6 и добавяне на получените F-зависимости към F, до тогава докато се получават нови (не принадлежащи на F) зависимости.

Нека F={ AB→C, C→B} е множество от F-зависимости за релацията r(A, B, C). F^+ = {A →A, AB→A, AC→A, ABC→A, B→B, AB→B, BC→B, ABC→B, C→C, AC→C, BC→C, ABC→C, AB→AB, ABC→AB, AC→AC, ABC→AC, BC→BC, ABC→BC, ABC→ABC, AB→C, AB→AC, AB→BC, AB→ABC, C→B, C→BC, AC→B, AC→AB}.

Тук и по-нататък не се разглеждат зависимости от вида X→∅ и ∅→X.

Намирането на обвивката F^+ на едно множество от функционални зависимости в общия случай е трудоемка задача. Причина за това е големият брой на елементите на обвивката. Въпреки това, тъй като процесът може да се автоматизира, извеждането на нови F-зависимости е много полезно за изследване на дадена релационна схема. Съществуват редица методи за получаване на F^+.

На практика по-често се налага да се решава по-проста задача за определяне дали дадена F-зависимост принадлежи на F^+. За решаването на тази задача също съществуват различни методи. Най-подходящ за автоматизиране е методът, при

който най-напред се определя **обвивка на атрибут**. Множеството $\mathbf{X^+}$ се нарича обвивка на атрибута X, ако за всяко Y, $Y \subseteq A$, за което $X \rightarrow Y$ е F-зависимост, следва че $Y \subseteq X^+$. С други думи, X^+ е подмножество Y на A с най-много елементи (максимален атрибут), за който $X \rightarrow Y$.

Алгоритъм за определяне на X^+

Вход: Атрибут X и множество от F-зависимости F.

Изход: Обвивка на X над F.

```
Closure (X, F)
begin
    OldDep:=∅; NewDep:=X;
    while NewDep ≠ OldDep do
        begin
            OldDep:=NewDep;
            for всяка F-зависимост W→Z от F do
            if NewDep ⊇ W then NewDep:= NewDep∪Z
        end
    return(NewDep)
end.
```

Използвайки Closure лесно се проверява дали $X \rightarrow Y$ принадлежи на F^+.

Алгоритъм за проверка

Вход: Множество от F-зависимости F и F-зависимост $X \rightarrow Y$.

Изход: True, ako $X \rightarrow Y$ принадлежи на F, False - в противен случай.

```
Member (F, X→Y)
begin
    if Y ⊆ Closure (X, F) then return(True) else return(False)
end.
```

6.4.2 Покрития

Множествата от F-зависимости могат да бъдат представени по различен начин, т.е. да имат различен брой и различни елементи, но техните обвивки да съвпадат. Например, всяка F-зависимост, която може да се изведе от множеството $F=\{A\rightarrow B, B\rightarrow C, A\rightarrow C, AB\rightarrow C, A\rightarrow BC\}$ може да бъде изведена и от множеството $G=\{A\rightarrow B, B\rightarrow C\}$, тъй като всички F-зависимости от F могат да се изведат от F-зависимостите на G. Възниква въпросът кое от двете множества е за предпочитане. Обикновено "по-малкото" множество от F-зависимости гарантира по-бързо изпълнение на алгоритмите.

Казваме, че две множества от F-зависимости F и G са еквивалентни ако имат една и съща обвивка, т. е. $F^+ = G^+$. Казва се още, че F покрива G или G покрива F.

Разглеждането на еквивалентните множества се налага от факта, че всеки две еквивалентни множества от F-зависимости са носители на едни и същи семантични знания за съответната релационна схема. В такъв случай особено важно за проектирането на релационната схема е да се намери "най-простото" множество от F-зависимости, което покрива първоначалното.

Основавайки се на факта, че $F^+ = G^+$, тогава и само тогава когато

$$f \in F \rightarrow f \in G^+$$

и $$g \in G \rightarrow g \in F^+$$

може да се формулира следния алгоритъм за проверка еквивалентността на две множества от F-зависимости:

Вход: Две множества от F-зависимости F и G.
Изход: True, ako F и G са еквивалентни, False - в противен случай.

```
Equiv (F, G)
begin
        v:=True;
        for всяка F-зависимост X→Y от G do
                v:=v and Member(F, X→Y);
        for всяка F-зависимост X→Y от F do
                v:=v and Member(G,X→Y);
        return(v)
end.
```

Едно множество от F-зависимости F се нарича **покритие без излишество**, ако никое собствено подмножество на F не е еквивалентно на F. Например, множеството от F-зависимости G={A→B, B→C, AB→C, AC→B} е покритие с излишество на множеството от F-зависимости F={A→B, B→C}, тъй като F⊂G и $F^+ = G^+$.

Интересно е да се отбележи, че има случаи когато едно множество F-зависимости има няколко покрития без излишество.

Едно множество от F-зависимости е **минимално покритие**, ако то не съдържа повече F-зависимости от кое да е, еквивалентно на него, множество от F-зависимости. За всяко множество от F-зависимости съществува минимално покритие.

7. Нормализация. Нормални форми

7.1 Основни положения

При работа с релационни схеми се използват свойствата и особеностите на ключовите атрибути, които са подмножество на атрибутите на дадена релация. Стойностите на ключа еднозначно определят редовете на релацията. С други думи, ако K е ключ на релация с релационна схема R(A), то в релацията не съществува нито една двойка редове, за която $t_1.K = t_2.K$.

Отчитайки определението на понятието функционална зависимост, можем да видим, че ключ на релационната схема R(A) е такова подмножество K ($K \subseteq A$), за което $K \rightarrow A$ и никое подмножество $K' \subseteq K$ не притежава това свойство. Следователно, ако в релацията съществуват два реда, такива че $t_1.K = t_2.K$, то непременно $t_1 = t_2$.

В общия случай ключът на една релация е съставен атрибут. Един атрибут X от релационната схема R(A), $X \in A$, се нарича **първичен атрибут**, ако влиза в състава на ключа, т.е. ако е част от ключа. В противен случай X е **непървичен атрибут**.

Често при разглеждане на функционалните зависимости, които дадена релация удовлетворява, се различават пълни и частични функционални зависимости. Атрибутът Y се намира в **пълна функционална зависимост** от атрибута X, $X \subseteq A$, $Y \subseteq A$, ако $\{X \rightarrow Y\} \in F^+$ и за всяка друга F-зависимост $X' \rightarrow Y$, $X' \subseteq X$ е изпълнено $\{X' \rightarrow Y\} \notin F^+$. В противен случай функционалната зависимост $X \rightarrow Y$ е **частична**.

7.2 Нормални форми

Нормализацията на една релационна схема е процес, при който схемата се преобразува с цел новополучените схеми да бъдат наложени известни ограничения като се елиминират някои нежелани свойства (аномалии от излишество, добавяне, отстраняване, актуализация).

7.2.1 Първа нормална форма (1НФ, ПНФ)

Релационната схема R(A) се намира в **първа нормална форма** ако всички атрибути от релационната схема имат атомарен характер, т.е. стойността на такъв атрибут се разглежда винаги само като една стойност и

никога не се разделя за да се използват отделни части от тази стойност. Например, атрибутът ЕГН се приема като атомарен, ако винаги се разглежда като една цяла стойност. Ако е необходимо да се разпознават ден, месец и т.н. от стойността на ЕГН, то ЕГН вече не се разглежда като атомарен атрибут. Ако е необходимо от стойността на ЕГН да се разпознават отделни части като самостоятелни стойности и атрибутът ЕГН е атрибут на дадена релационна схема, то тази релационна схема не е в ПНФ.

Атомарността е понятие, което се определя във всеки конкретен случай.

Схемата на една БД се намира в ПНФ, ако всички релационни схеми, които влизат в състава и са в ПНФ.

Преимуществата на ПНФ са свързани с възможността да се изразяват F-зависимостите с такава степен на детайлизиране, каквато е необходима по преценка на разработчика във всеки конкретен случай. Например, дали е необходимо да се използват поотделно данните за ден, месец и година на раждане, или е достатъчно да се разглежда неделим атрибут ЕГН, който на практика съдържа горните данни, се определя при разработване на самата БД.

На практика тази дефиниция на ПНФ противоречи в известен смисъл на определението за релация. Но, тъй като исторически се е наложило нормализирането да се извършва чрез преобразуване от ПНФ към нормални форми от по висок ред, ще обсъдим ПНФ за пълнота.

Ненормализираната релация можем да разглеждаме още като таблица с неравномерно запълване. Ще разгледаме таблица “Разпис” , която има следния вид:

Преподавател	**Ден**	**Време**	**Дисциплина**	**Вид на занятието**	**Група**
Тодоров	Понеделник	9:15	ООП	Лекция	43
	Вторник	9:15	УП	Лекция	42
	Вторник	11:15	СДП	Сем. упр.	43
Кирова	Вторник	13:15	КГ	Лекця	42
	Сряда	13:15	КГ	Сем. упр.	39
	Петък	9:15	КГ	Пр. упр.	39
	Петък	11:15	КГ	Пр. упр.	39
Стоев	Четвъртък	11:15	ООП	Пр. упр.	43
	Четвъртък	14:15	ООП	Пр. упр.	43

Тук, при пресичането на един ред и една колона, се срещат повече от една елементарни стойности, съответстващи на броя на дните, в които един преподавател има занятия. За привеждане на релацията в ПНФ е необходимо да се увеличат редовете, така че всяка клетка от таблицата да съдържа само една стойност:

Преподавател	Ден	Време	Дисциплина	Вид на занятието	Група
Тодоров	Понеделник	9:15	ООП	Лекция	43
Тодоров	Вторник	9:15	УП	Лекция	42
Тодоров	Вторник	11:15	СДП	Сем. упр.	43
Кирова	Вторник	13:15	КГ	Лекция	42
Кирова	Сряда	13:15	КГ	Сем. упр.	39
Кирова	Петък	9:15	КГ	Пр. упр.	39
Кирова	Петък	11:15	КГ	Пр. упр.	39
Стоев	Четвъртък	11:15	ООП	Пр. упр.	43
Стоев	Четвъртък	14:15	ООП	Пр. упр.	43

7.2.2 Втора нормална форма (2НФ, ВНФ)

Едно отношение в първа нормална форма удовлетворява редица функционални зависимости. В общия случай за такова отношение са налице аномалиите от излишество, добавяне, отстраняване и актуализация. Тези аномалии са свързани с F-зависимостите, които отношението удовлетворява. Някои от непървичните атрибути на едно отношение са в пълна зависимост от ключовия атрибут, а други - само в частична зависимост. Може да се покаже, че част от аномалиите се дължат именно на такива частични зависимости. Недостатъците на една релационна схема, в която съществуват частични F-зависимости на непървичните атрибути от първичните, могат да се отстранят като тази релационна схема се замени с две или повече релационни схеми, в които частичните зависимости от ключа са премахнати.

Казва се, че една релационна схема е във **втора нормална форма** по отношение на множеството от F-зависимости F, ако тя е в ПНФ и всеки непървичен атрибут е в пълна функционална зависимост от ключа на релацията.

Схемата на една БД се намира във ВНФ, ако всички релационни схеми, които влизат в състава и са във ВНФ.

Ще разгледаме релация, която представя резултатите на студентите от текущата сесия. Релацията има следната схема:

Изпит (Фамилия, Фак. номер, Група, Дисциплина, Оценка)

Тъй като един студент се явява на повече от един изпит през сесията, то първичен ключ на релацията може да бъде ***(Фак.номер, Дисциплина),*** защото те заедно еднозначно определят всеки ред от тази релация.

Можем да забележим, че атрибутите ***Фамилия*** и ***Група*** зависят само ***Фак. номер***, т.е. от част от първичния ключ. Това означава, че има частични зависимости в релационната схема. За привеждане на релационната схема във ВНФ е необходимо да разбием тази схема на две, така че възстановяването на изходното отношение да е възможно от новополучените. Ще направим това по следния начин:

Студент (Фамилия, Фак. номер, Група)
Изпит (Фак. номер, Дисциплина, Оценка)

Този набор от релационни схеми не съдържа частични зависимости, а само пълни, затова всяка от тях е във ВНФ.

Защо е необходимо привеждането на релационната схема във ВНФ? Какви аномалии или неудобства могат да възникнат ако не приведем релационната схема във ВНФ? Нека да разгледаме ситуация, при която студент се премества от една група в друга. Тогава, ако релационната схема не беше приведена във ВНФ, е необходимо да се открият всички редове, в които има информация за този студент, и да се промени стойността на атрибута ***Група***. Във втория случай ще е необходимо да се промени стойността на същия атрибут само на едно място. Естествено, във втория случай, възможността за нарушаване на непротиворечивостта е много по-малка. В първия случай би могло да се случи така, че по време на коригиране на стойността на атрибута Група на повече от едно място, заради машинен сбой например, да се окаже, че в различни редове на изходната релация, този студент ще бъде причислен към различни групи. Ако релационната схема е във ВНФ, ще се наложи промяната да се извършва само на едно място.

Освен това, ако има студенти, които все още не са се явявали на изпит, то за тях е невъзможно да се въведе информация в първия вариант на БД, докато във втория случай информацията за студентите се съхранява отделно от резултатите от явяването им на изпити.

7.2.3 Трета нормална форма (3НФ, ТНФ)

Една релационна схема, която е във ВНФ, има по-добри свойства от релационна схема в ПНФ. Въпреки това, някои аномалии са налице и в релации с такава релационна схема. Причина за това е наличието на транзитивни функционални зависимости. Нека R(A) е релационна схема със съответно множество от функционални зависимости F. Казва се, че атрибутът Z е транзитивно зависим от атрибута X, ако съществува такова подмножество от атрибути Y, $Y \subset A$ и $Z \notin XY$, за което е изпълнено

$$[X \rightarrow Y\} \in F^+$$

$$\{Y \rightarrow X\} \notin F^+$$

$$\{Y \rightarrow Z\} \in F^+$$

Една релационна схема R(A) е в **трета нормална форма** по отношение на множеството от F-зависимости F, ако тя е в ПНФ и нито един непървичен атрибут не е транзитивно зависим от ключа. Недостатъците на една релационна схема, в която съществуват транзитивни F-зависимости на непървичните атрибути от първичните, могат да се отстранят като тази релационна схема се замени с две или повече релационни схеми, в които транзитивните зависимости от ключа са премахнати.

Схемата на една БД се намира в ТНФ, ако всички релационни схеми, които влизат в състава и са в ТНФ.

В определението за ВНФ се изисква наличие на ПНФ, т.е. всяка релационна схема, която е във ВНФ, е и в ПНФ. В определението за ТНФ не се изисква релационната схема да е във ВНФ, но може да се докаже, че всяка релационна схема, която е в ТНФ, е и във ВНФ.

Да допуснем, че релационната схема R(A), е в ТНФ, но не е във ВНФ. Нека Z, $Z \subset A$, не е в пълна функционална зависимост от ключа К на R(A). Това означава, че съществува подмножество К' на К, за което е в сила F-

зависимостта K'→Z. Но тъй като K е ключ, следва че {K→K'}∈F^+и {K'→K}∉F^+, т.е. налице е транзитивна зависимост, което противоречи на направеното по-горе предположение.

Ще разгледаме релационна схема, представяща връзките между студенти, групи, факултети и специалности.

Студент (Фамилия, Фак. номер, Група, Факултет, Специалност, Катедра)

Първичен ключ на тази релационна схема се явява ***Фак. номер***, но ще разгледаме и другите функционални зависимости:

Фак. номер → Фамилия

Фак. номер → Група

Фак. номер → Факултет

Фак. номер → Специалност

Фак. номер → Катедра

Група → Факултет

Група → Специалност

Група → Катедра

Катедра → Факултет

Сред тези зависимости има такива, които образуват транзитивни групи. За да избегнем тяхното влияние, можем да преобразуваме релационната схема в следния набор от релационни схеми:

Студент (Фамилия, <u>Фак. номер</u>, Група, Специалност)

Група (<u>Група</u>,Катедра)

Катедра (<u>Катедра</u>, Факултет)

Първичните ключове на всяка релационна схема са подчертани.

Можем да забележим, че в нито една от тези релационни схеми няма транзитивни зависимости.

7.3 Нормализация чрез декомпозиция

Винаги е възможно една релационна схема, която не е в ТНФ по отношение на множество от F-зависимости F да се разложи на няколко релационни схеми, всяка от които е в ТНФ по отношение на F. Разлагането на една релационна схема R(A) е замяна на тази схема с две схеми P и Q (обикновено пресичащи се), така че всяка релация с релационна схема R(A), удовлетворяваща F, да може да се разложи на релации с релационни схеми P(X) и Q(Y), $X \cup Y \equiv A$, без загуба на информация. С други думи, новите релационни схеми имат за атрибути някои от атрибутите на R, като обединението им дава множеството от атрибути A, а сечението им може да е както празно, така и непразно множество. Ако някоя от релационните схеми P и Q не в ТНФ, то процесът може да се повтори спрямо тази релационна схема. Процесът продължава докато всички получени релационни схеми са в ТНФ по отношение на множеството F.

Нека R(A, X, B) е дадена релационна схема, а R* е нейната декомпозиция R* = \ P(AX), Q(XB)\. Предполага се ,че:

1. r релация с релационна схема R(A, X, B)
2. p = π_{AX}(r), т.е. p: P(AX)
3. q = π_{XB}(r), т.е. q: Q(XB)

Естествено е да се зададе въпросът, дали при така формулираните предположения може да се твърди, че релацията r е възстановима по нейните проекции p и q.

Нека s= p $\triangleright\triangleleft$ q. т.е. s е естествено съединение на p и q по отношение на атрибута X. Ако r=s , за всяко r с релационна схема R(A) и съответно множество от функционални зависимости F, тогава се казва, че декомпозицията на r притежава свойството **съединение без загуба по отношение на множеството F**.

Нека R(A) е релационна схема с множество от функционални зависимости F и R* = { P(X), Q(Y)} е една нейна декомпозиция. Ако r е релация с релационна схема R(A), p е релация с релационна схема P(X), а q е релация с релационна схема Q(Y), тогава

1. $r \subseteq p \triangleright\triangleleft q$
2. r = p $\triangleright\triangleleft$ q, тогава и само тогава, когато

$$\{ X \cap Y \rightarrow X\text{-}Y\} \in F^{+} \quad \text{или} \{ X \cap Y \rightarrow Y\text{-}X\} \in F^{+}$$

Същността на декомпозицията се заключава в преобразуване на релационни схеми, които не са в ТНФ по отношение на дадено множество от функционални зависимости F, в релационни схеми, които са в ТНФ. Това преобразуване се извършва на две стъпки. В първата от тях релационните схеми се привеждат във ВНФ, а във втората стъпка - в ТНФ. Ще разгледаме най-напред идеята за привеждането на релационните схеми във ВНФ.

Нека R(A) е релационна схема, която е в ПНФ, но не е във ВНФ. Това означава, че:

1. ключът на тази релационна схема е съставен, т.е. той се състои поне от два атрибута - $К_1$ и $К_2$, $К_1 \in A$ и $К_2 \in A$;

2. съществуват две подмножества на A, X и Y, $X \subset A$ и $Y \subset A$, такива че $К_1К_2 \rightarrow Y$ и $К_1 \rightarrow X$, т.е. атрибутът X не се намира в пълна функционална зависимост от ключа (предполагаме, че $A=\{К_1К_2XY\}$).

Възможно е релационната схема $R(К_1К_2XY)$, която не във ВНФ, да се замени с две релационни схеми $P(К_1К_2Y)$ и $Q(К_1X)$. Ако P и Q не са във ВНФ, то описаната процедура се прилага отново. Това продължава докато се получи множество от релационни схеми, всички от които са във ВНФ.

За привеждане на релационните схеми в ТНФ се прилага следната идея.

Нека R(A) е релационна схема с множество от F-зависимости F, която е във ВНФ, но не е в ТНФ. Това означава, че в R(A) има транзитивни функционални зависимости, т.е. може да се приеме, че множеството от атрибути A се състои от K, X, Y и Z, където K е ключ, а X, Y и Z са неключови атрибути, за които $\{K \rightarrow X\} \in F^+$ и $\{X \rightarrow Y\} \in F^+$. Възможно е атрибутът Z да не съществува.

При тези предположения първоначалната релационна схема R(K, X, Y, Z) може да се замени с две релационни схеми P(K, X, Z) и Q(X, Y).

В общия случай сложността на този алгоритъм е твърде голяма. Това е така защото в резултат от декомпозицията може да се получи много по-голям брой релационни схеми в ТНФ, отколкото чрез прилагане на други методи.

7.4 Синтез на релационни схеми

Основна цел и на този метод е получаване на схема на БД като набор от релационни схеми, които са в ТНФ по отношение на известно множество от

функционални зависимости, т.е. това е метод за синтез на нормализирани релационни схеми

Идеята на метода за синтез е следната:

1. Разглежда се множеството от функционални зависимости F, като отделните зависимости се групират по равенство на левите им страни.

2. За всяка група, получена в т.1, се дефинира отделна релационна схема с множество от атрибути, участващи във функционалните зависимости на всяка отделна група.

Не всяка от получените по този начин релационни схеми, обаче, ще бъде в ТНФ. Това се дължи на наличието на излишни атрибути по отношение на някои от F-зависимостите и излишни F-зависимости в F. Понятието излишна F-зависимост е вече известно, ще определим и понятието излишен атрибут.

Нека функционалната зависимост $X \rightarrow Y$ е от F. Атрибутът X се нарича атрибут без излишество по отношение на функционалната зависимост $X \rightarrow Y$, ако Y се намира в пълна функционална зависимост от X.

За да се извърши успешно синтезиране на нормализирани релационни схеми е необходимо предварително да се отстранят както излишните атрибути, така и излишните F-зависимости. Излишните F-зависимости се отстраняват с помощта на алгоритъма за намиране на покритие без излишество, който вече е известен.

Алгоритъм за отстраняване на излишни атрибути:

Вход: F-зависимост $X' \rightarrow Y$ и множество от F-зависимости F.

Изход: F-зависимост $X \rightarrow Y$, която не съдържа излишни атрибути.

```
Nonredun_attr(X', Y)
begin
    X:=X';
    for всеки атрибут Z от A do
        if Z∈X then begin
                        W:=X - Z;
                        if Member({W→Y}, F) then X:=X - Z;
                    end;
    return(X)
end.
```

Прилагайки този алгоритъм към всяка F-зависимост от F, можем да заменим F-зависимостите, които съдържат излишъци в левите си части с други F-зависимости без излишъци.

След тези трансформации може да се приложи следния алгоритъм за синтез на релационни схеми, които са в ТНФ:

1. Прилага се алгоритъма за отстраняване на излишните атрибути за всяка F-зависимост от F.

2. Прилага се алгоритъма за намиране на покритие без излишество на F.

3. Разделя се множеството от F-зависимости на групи, всяка от които съдържа F-зависимости с еднакви леви части.

4. От всяка група F-зависимости се определя релационна схема, която съдържа обединението на атрибутите, фигуриращи в лявата и дясната част на F-зависимостите от една група. За ключове на релационните схеми се избират атрибутите, които са в левите части на функционалните зависимости от всяка група.

8. Управление на транзакциите

8.1 Общи положения

Различните СУБД предлагат на потребителите използване на базите от данни в режим на работа на **един потребител (Single_User Systems)** или в режим на работа на **много потребители (Multiuser Systems)**. При работа в режим на един потребител транзакциите се изпълняват последователно една след друга. При работа в многопотребителски режим е възможно едновременно изпълнение на транзакциите, представляващи заявките на различните потребители.

Заявките на различните потребители могат да съдържат действия както с данни от различни БД, така и други действия. Всяка заявка предварително се преобразува в поредица от логически самостоятелни действия - **транзакции (trnasactions)**, които трябва да се извършат, за да се изпълни заявката.

Транзакцията е логическа единица работа. Например, ако е необходимо да се промени кодът на доставчика от Sx на Sy, за потребителя това се възприема като едно действие. За дадена БД, обаче, тази задача може да изисква извършването на повече от едно обновяване, тъй като атрибутът ***Код_на _доставчика*** може да присъства в няколко отношения. Във времето между началото на първото обновяване и края на последното БД ще съдържа противоречиви данни. Така че, всяка транзакция може да се изпълнява като последователност от действия в БД, за да се трансформира тя от едно непротиворечиво състояние в друго такова. Ясно е, че не бива да се допускат други действия, засягащи данните от БД, които дадена транзакция обработва, преди да завърши изпълнението на цялата транзакция. За да се осигури това, са нужни специални средства в СУБД. Информацията трябва да бъде защитена не само от междинни опити за достъп до нея. Възможно е преди да е завършила транзакцията да се получи някакво аварийно прекъсване на работата на системата. Затова, ако една транзакция не е завършила нормално, информацията в БД се запазва в състояние, каквото е имала преди началото на транзакцията. Предвиждат се специални средства, които проверяват и съобщават как е била изпълнена дадена транзакция - дали е завършила успешно или трябва състоянието на БД да се възстанови до първоначалното (при неуспешно изпълнение на дадена транзакция). Възстановяването се организира като, преди извършването на някакви действия с данните, тяхното състояние се запазва и може да се възстанови при неуспешна транзакция.

8.2 Свойства на транзакциите. Начини на завършване

За нормална работа с данните от БД всяка транзакция е желателно да притежава следните свойства:

1. Атомарност (Atomicity) - транзакцията трябва да представлява самостоятелно, логически завършено действие с данните от БД. Тя се изпълнява изцяло или не се изпълнява въобще.

2. Непротиворечивост (Consistency preservation) - коректното изпълнение на транзакцията трябва да преобразува БД от едно непротиворечиво състояние в друго, т.е транзакцията не разрушава непротиворечивото състояние на БД.

3. Изолираност (Isolation) - промените, които дадена транзакция извършва с данните от БД не трябва да бъдат "видими" за останалите транзакции докато тази транзакция не е завършила. На практика тези конкуриращи се за достъп към БД транзакции се изпълняват последователно, но за потребителя това изглежда като едновременна обработка.

4. Устойчивост (Durability) - след като дадена транзакция е завършила успешно, промените, направени от нея в БД, остават постоянни.

Възможни са два начина на завършване на транзакциите. Ако всички операции са изпълнени успешно и, по време на изпълнение на транзакцията не имало аварийно прекъсване на работата, транзакцията се фиксира.

Фиксиране на транзакцията е действие, което осигурява запис на дисков носител на настъпилите, в резултат на изпълнения на транзакцията, изменения в БД. Докато транзакцията не е фиксирана е допустимо анулиране на направените изменения и връщане на състоянието на БД до такова, каквото тя е имала преди началото на транзакцията. Фиксирането означава, че всички резултати от нейното изпълнение стават постоянни. Те стават видими за другите транзакции след фиксирането на транзакцията. Преди да настъпи моментът на фиксирането, данните, които дадената транзакция обработва, за останалите транзакции имат стойности, каквито са имали преди започването на разглежданата транзакция.

8.3 Некоректно изпълнение на транзакциите. Възстановимост

При разглеждане на изпълняваните действия с данните в БД винаги е необходимо да се отчита това, че данните реално са разположени върху външни носители, а действията с тях могат да се извършват само в ОП. Ето защо винаги се извършва прехвърляне на данните от външната памет в ОП и обратно.

При изпълнение на всяка транзакция системата трябва да се грижи за:

- успешното изпълнение на всяка транзакция и записването на резултата от нейното изпълнение върху носителя, на който се съхраняват данните от БД;

- разрешаване на изпълнението на дадена транзакция само в случай, че съществуват необходимите условия за това. Възможно е някаква предшестваща транзакция да не е завършила успешно, което да прави безсмислено изпълнението на друга транзакция.

Причини за неуспешно завършване на транзакциите:

1. Компютърна повреда.

В такъв случай може да се получи физическа повреда на носителя, на който е разположена БД.

2. Грешка по време на изпълнение на самата транзакция или програма от СУБД - деление на нула, препълване и др.

3. Невъзможност да се изпълни транзакцията - липса на необходимите данни в БД и др.

4. Невъзможност да се изпълни транзакцията при многопотребителска работа. Системата забранява временно изпълнението на дадена транзакция, тъй като други транзакции работят със същите данни, не е изпълнена успешно предна транзакция и др.

5. Повреда на устройството, на което е носителя с данните от БД или повреда на самия носител.

Всяка СУБД трябва да разполага със средства и начини за възстановяване на изпълнението на потребителските заявки след възникването на всяка от горните ситуации.

За случай 5. всяка система обикновено предлага функции DUMP и RESTORE, с помощта на които периодично се съхранява някакво състояние на БД на резервен носител. При необходимост се възстановява от него.

За ситуациите 1. - 4. СУБД обикновено поддържат специални **журнални (log) файлове**, в който се съхранява помощна информация за последователността и резултата от изпълнението на всички транзакции.

Журналният файл съдържа данни както за времето на започване и времето на приключване на всяка транзакция, така и за операциите Read и Write, които транзакцията евентуално включва. Отделните записи на този файл обикновено съдържат:

- име на транзакцията;

- име на елемента данни, който транзакцията обработва;

- стара стойност на тези данни;

- нова стойност на тези данни;

Допустими са и други полета в този файл, които са нужни за съхраняване на друга специфична информация, като начало на транзакцията, край на транзакцията и др.

Примери за записи в log файла:

$<T_i$ start>	Ti започва
$<T_i, X_j, V_1, V_2>$	Ti обновява елемента данни X_j като заменя стойността V_1 на V_2
$<T_i$ commit>	Ti приключва успешно
$<T_i$ end>	Ti завършва

Периодично журналният файл се съхранява върху диск **(checkpoint time)**. Checkpoint time се използва при определяне кои транзакции да се повторят след възникване на някаква грешка. Обикновено това са транзакциите, които са се изпълнявали след последното съхраняване.

На следващата фигура:

T_1 е завършила преди t_c

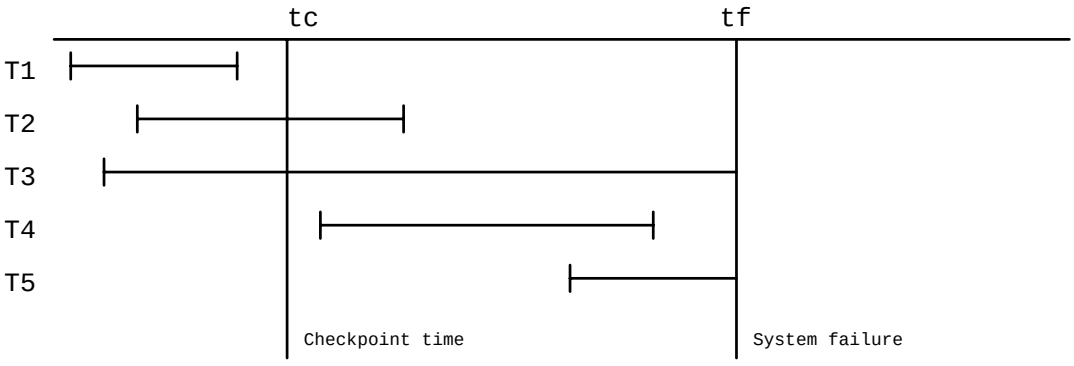
T_2 е завършила преди t_f

T_3 не е завършила

T_4 е завършила преди t_f

T_5 не е завършила

В този случай T_1 е завършила и резултатът от нея е съхранен в БД, T_3 и T_5 ще бъдат обявени като незавършени и трябва да се повторят, а T_2 и T_4 трябва да се повторят, тъй като резултатите от тях не са съхранени.



8.4 Едновременно изпълнение на транзакции. Графици

При работа на много потребители е възможно "едновременно" изпълнение на транзакции, принадлежащи на различни заявки. Редът на изпълнение на транзакциите и съставляващите ги операции се представя чрез т.н. **график (schedule or history)**. Включването на транзакциите в графици с определени характеристики позволява да се улесни възстановяването след грешки.

Да разгледаме транзакциите:

T1		T2
		Read(X)
	Read(X)	
X:=X-N		X:=X+M
	Write(X)	
Read(Y)		
Y:=Y+N		
Write(Y)		

Два възможни графика за изпълнение на тези транзакции са следните:

T1	T2
Read(X)	
X:=X-N	
	Read(X)
	X:=X+M
Write(X)	
Read(Y)	
	Write(X)
Y:=Y+N	
Write(Y)	

T1	T2
Read(X)	
X:=X-N	
Write(X)	
	Read(X)
	X:=X+M
	Write(X)
Read(Y)	
Y:=Y+N	
Write(Y)	

Когато няколко транзакции се изпълняват едновременно, непротиворечивостта на БД може да бъде нарушена въпреки правилното изпълнение на всяка от транзакциите. Ето защо е необходимо да се осигури такова изпълнение, при което непременно да се съхрани непротиворечивостта. Основно внимание от тази гледна точка се отделя на операциите Read и Write.

Графици, при които всяка транзакция започва само след като е завършила предходната, се наричат **серийни или последователни (serial)**. Всеки друг график се нарича несериен. Ясно е, че за N транзакции има N! различни серийни графици.

Смята се, че изпълнението на кой да е сериен график не нарушава непротиворечивостта на БД, тъй като всяка транзакция притежава това свойство и изпълнението на всяка транзакция е независимо от изпълнението на друга транзакция.

Серийните графици, обаче, на практика противоречат на едновременното изпълнение на заявките при многопотребителска работа. Ето защо е по-добре да се изпълняват несерийни графици, но до коректен резултат водят само някои от тях.

Да разгледаме следните графици:

График А

T1	T2
Read(X)	
X:=X-N	
Write(X)	
Read(Y)	
Y:=Y+N	
Write(Y)	
	Read(X)
	X:=X+M
	Write(X)

График Б

T1	T2
Read(X)	
X:=X-N	
	Read(X)
	X:=X+M
Write(X)	
Read(Y)	
	Write(X)
Y:=Y+N	
Write(Y)	

График В

T1	T2
Read(X)	
X:=X-N	
Write(X)	
	Read(X)
	X:=X+M
	Write(X)
Read(Y)	
Y:=Y+N	
Write(Y)	

Нека началните стойности на елементите на данните са X=90 и Y=90 и N=3 и M=2. След изпълнението на график А: X=89 и Y=93, на график Б: X=92 и Y=93, на график В: X=89 и Y=93.

Несериен график, след изпълнението на който се получава резултат, равен на резултат от кой да е сериен график, се нарича **сериализуем (serializable)**. Изпълнението на всеки сериализуем график дава коректен резултат, тъй като той дава резултат, равен на резултата от даден сериен график, а вече е известно, че резултатите от серийните графици са коректни.

Съществуват различни начини за определяне на сериализуемостта на даден график. Един такъв начин вече разгледахме. Ще разгледаме и друг начин. За тази цел най-напред ще определим понятието **конфликтни операции (conflict operations)**.

Нека Oi и Oj са от Ti и Tj. Ако Oi и Oj се отнасят за различни елементи на данните, местата им в даден график могат да бъдат разменени без да се наруши серийността на този график. Ако се отнасят за един и същ елемент на данните:

- и двете са Read - редът им в графика е без значение;

- едната е Read, а другата Write - редът им в графика има значение;

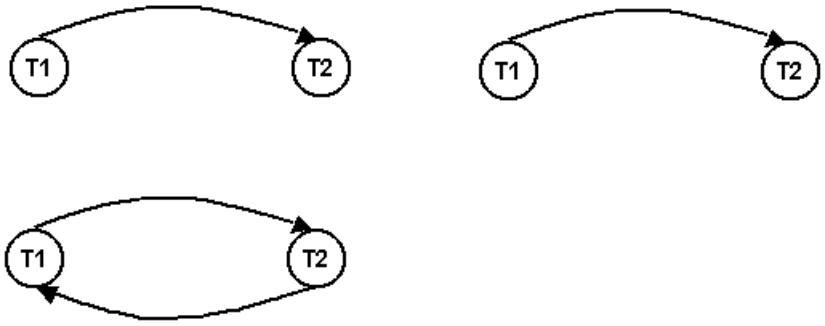
- и двете са Write - редът им в графика има значение, тъй като ще се запомни само втората стойност и това може да повлияе некоректно на следващите действия с БД.

Казваме, че две операции са конфликтни, ако са от различни транзакции, отнасят се един и същ елемент на данните и поне едната от тях е Write. Ако две конфликтни операции се появяват в различен ред в два графика, то резултатите от

тези два графика се очаква да бъдат различни. Това заключение се използва са като условие за проверка на сериализуемостта на даден график.

Проверка за сериализуемост на даден график може да се извърши и като се построи **граф на предхождане (precedence graph)**. Графът на предхождане е граф, в който на всяка транзакция съответства връх, а два възела, съответстващи на Ti и Tj се свързват с насочена дъга Ti-->Tj, ако операция от Ti се среща в графика преди нейна конфликтна операция от Tj. Графикът, съответстващ на такъв граф, е сериализуем, ако графът не съдържа цикли.

На следващата фигура са дадени графите на предхождане за графиците А, Б и В:



9. Управление на конкурентно изпълнение на транзакциите

9.1 Общи положения

Многопотребителското използване на БД предполага възникването на ситуации, в които едни и същи данни ще се използват от различни потребители. Всяка СУБД, която допуска такъв режим на работа, трябва да разполага със средства за управление на едновременния достъп до едни и същи данни и за разрешаване на конфликтните ситуации.

Съществуват различни методи за това. Повечето от тях имат за цел да осигурят сериализуемостта на даден график, който се изпълнява. Тези методи се основават на съвкупност от правила, спазването на които осигурява сериализуемостта.

9.2 Locking

Това е един от най-често използваните методи за управление на едновременния достъп на няколко транзакции до едни и същи данни. Всеки елемент от данните се свързва със специална променлива - Lock, стойността на която показва дали е възможно изпълнението на дадена операция с тези данни в даден момент.

9.2.1 Видове Locks

Binary Locks - могат да имат две стойности - 1, съответстваща на състояние locked, и 0 - съответстваща на състояние unlocked. Ако Lock(X)=1, то елементът данни X е недостъпен, а ако Lock(X)=0, то X може да се използва от текущата транзакция.

При използване на метода Locking с Binary Locks, към операциите, които съставляват отделните транзакции се добавят две операции - lock_item и unlock_item. Преди използването на елемента X се изпълнява операция lock_item(X) и след завършване на работата с тези данни - unlock_item(X). Ако дадена транзакция се опита да изпълни операция lock_item(X) когато Lock(X)=1, тя не трябва да може да направи това. Тя трябва да изчака освобождаването на елемента X. Ако Lock(X)=0, то операцията lock_item го установява в 1 и изпълнението на транзакцията продължава със следващите операции.

Locking с Binary Locks изисква при изпълнението си всяка транзакция да следва правилата:

1. Изпълнява операция lock_item(X) преди коя да е операция read(X) или write(X).

2. Изпълнява операция unlock_item(X) след всички операции read(X) или write(X).

3. Не изпълнява повторно операция lock_item(X).

4. Не изпълнява unlock_item(X) преди да е изпълнила операция lock_item(X).

За спазването на тези правила се грижи специален модул от СУБД - Lock Manager.

Multiple-mode Locks

В този случай се използват три допълнителни операции - read_lock(X), write_lock(X) и unlock(X). Променливата Lock(X) в този случай може да има три различни стойности. Операцията read-lock(X) се нарича още **shared lock**, тъй като след нейното изпълнение елементът данни X е достъпен за четене и от други транзакции (но само за четене). Операцията write_lock(X) се нарича още **exclusive lock**, тъй като тя забранява всякакъв достъп на друга транзакция до X.

В този случай всяка транзакция следва правилата:

1. Съдържа операция read_lock(X) или write_lock(X) преди коя да е операция read(X).

2. Съдържа операция write_lock(X) преди коя да е операция write(X).

3. Съдържа операция unlock(X) след всички операции read_lock(X) или write_lock(X).

4. Не изпълнява операция read_lock(X) след като вече е изпълнила операция read_lock(X) или write_lock(X).

5. Не изпълнява операция write_lock(X) след като вече е изпълнила операция read_lock(X) или write_lock(X).

6. Не изпълнява unlock(X) преди да е изпълнила операция read_lock(X) или write_lock(X).

Използването на binary и multiple-mode loks не винаги гарантира сериализуемостта на даден график.

Пример:

Дадени са транзакциите:

T_1	T_2
read_lock(Y)	read_lock(X)
read(Y)	read(X)

unlock(Y)	unlock(X)
write_lock(X)	write_lock(Y)
read(X)	read(Y)
X:=X+Y	Y:=X+Y
write(X)	write(Y)
unlock(X)	unlock(Y)

Начални стойности: X=20, Y=30

Резултат от сериен график, в който T_1 предшества T_2: X=50, Y=80

Резултат от сериен график, в който T_2 предшества T_1: X=70, Y=50

Графикът

T_1	T_2
read_lock(Y)	
read(Y)	
unlock(Y)	
	read_lock(X)
	read(X)
	unlock(X)
	write_lock(Y)
	read(Y)
	Y:=X+Y
	write(Y)
	unlock(Y)
write_lock(X)	
read(X)	
X:=X+Y	
write(X)	
unlock(X)	

дава резултат: X=50, Y=50.

9.2.2 Гарантиране на сериализуемост чрез Two-phase Locking

Казва се, че дадена транзакция следва t**wo-phase locking protocol** когато операциите lock_item, read_lock, write_lock предшестват първата unlock операция в транзакцията.

В горния пример транзакциите T_1 и T_2 не съблюдават това правило - операцията write_lock(Y) от T_2 се изпълнява след операцията unlock(X) от същата транзакция и операцията write_lock(X) се изпълнява след операцията unlock(Y) в T_1. За да удовлетворяват two-phase locking protocol транзакциите T1 и T2 трябва да се променят по следния начин:

T_1	T_2
read_lock(Y)	read_lock(X)
read(Y)	read(X)
write_lock(X)	write_lock(Y)
unlock(Y)	unlock(X)
read(X)	read(Y)
X:=X+Y	Y:=X+Y
write(X)	write(Y)
unlock(X)	unlock(Y)

Може да се докаже, че ако всяка транзакция в даден график удовлетворява two-phase locking protocol, то всеки график е сериализуем.

9.2.3 Deadlock

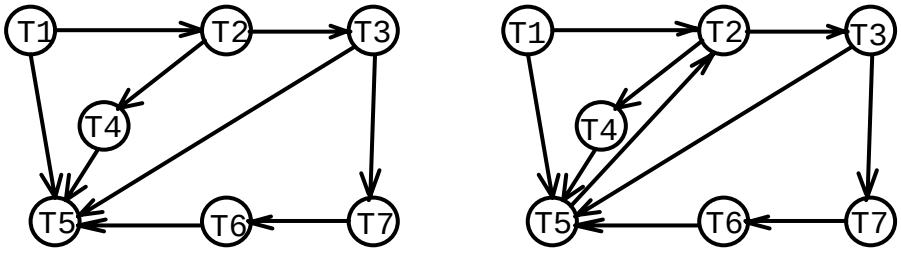
Ситуацията, при която всяка от две транзакции чака другата да освободи един и същ елемент от данни се нарича **deadlock**. В този случай двете транзакции взаимно блокират по-нататъшното си изпълнение. За продължение на нормалната работа на СУБД са необходими специални действия.

Пример:

T_1	T_2
read_lock(Y)	
read(Y)	
	read_lock(X)
	read(X)
write_lock(X)	
	write_lock(Y)

Съществуват начини за установяване на възникнал deadlock и прекратяването му. В общия случай deadlock възниква когато се получи затворена верига от

транзакции, всяка от които изчаква приключването на друга транзакция от веригата и освобождаване на нужния елемент от данните. Алгоритъмът за откриване на deadlock се състои в откриване на цикъл в **wait-for граф**. Wait-for граф е ориентиран граф, който съдържа възли, съответстващи на активните транзакции и насочени дъги, свързващи транзакция, която изчаква (начало на дъгата) и транзакция, която използва (край на дъгата) даден елемент от данните.



Нормално изпълнение на транзакциите T1-T7 Deadlock

При установяването на deadlock СУБД обикновено изчаква определено време, след което отново проверява дали това състояние продължава. Ако състоянието на deadlock продължава прекалено дълго, СУБД разрешава проблема като избира една от транзакциите, която прекратява и я обявява за неизпълнена. Като такава може да бъде избрана транзакция, която е започнала изпълнението си най-късно или такава, чието изпълнение продължава най-дълго и др.

Понякога, вместо начини за прекратяване на deadlock, се прилагат методи за предотвратяване на неговото възникване.

9.3 Timestamp ordering

9.3.1 Timestamps

Timestamp е уникален идентификатор, който СУБД създава за всяка транзакция. Timestamp получава стойност, съответстваща на реда на появяването на транзакцията за изпълни ние. Ще отбелязваме timestamp на транзакцията T с TS(T). Тъй като при този метод за управление на едновременно изпълнение на транзакциите не се използват locks, то не се появяват и deadlocks.

Timestamps се генерират по различни начини. Един възможен начин е да се използва брояч, чиято стойност се увеличава с единица при появяването на всяка нова транзакция. Друг начин е използването на часовника на компютъра, като се следи да не възникне ситуация, в която две транзакции да получат еднакви timestamps.

9.3.2 Управление на едновременното изпълнение на транзакциите чрез Timestamp ordering

Основнате идея на този метод е подреждането на транзакциите за изпълнение според техните timestamps. Полученият по този начин график е сериализуем (трябва да се отбележи, че могат да се получават сериализуеми графици, еквивалентни само на един последователен график). СУБД се грижи използването на даден елемент от данните от няколко транзакции да става по реда на техните timestamps и по този начин да съхранява сериализуемстта на изпълнявания график. Това става като на всеки елемент от данни X се съпоставят две стойности:

read_TS(X) - най-големият timestamp измежду timestamps на транзакциите, които успешно са прочели X;

write_TS(X) - най-големият timestamp измежду timestamps на транзакциите, които успешно са записали X.

Когато някоя транзакция T трябва да прочете или запише X, СУБД сравнява TS(T) с read_TS(X) или write_TS(X), за да се увери, че определеният ред за изпълнение на транзакциите не се нарушава. Извършват се проверки за следните два случая:

1. Транзакцията T трябва да изпълни write_item(X):

Ако read_TS(X)>TS(T) или ако write_TS(X)>TS(T), тогава изпълнението на T се прекратява.

Ако read_TS(X)<=TS(T) и write_TS(X)<=TS(T), тогава се изпълнява write_item(X) от T и write_TS(X) приема стойността на TS(T).

2. Транзакцията T трябва да изпълни read_item(X):

Ако write_TS(X)>TS(T), тогава изпълнението на T се прекратява.

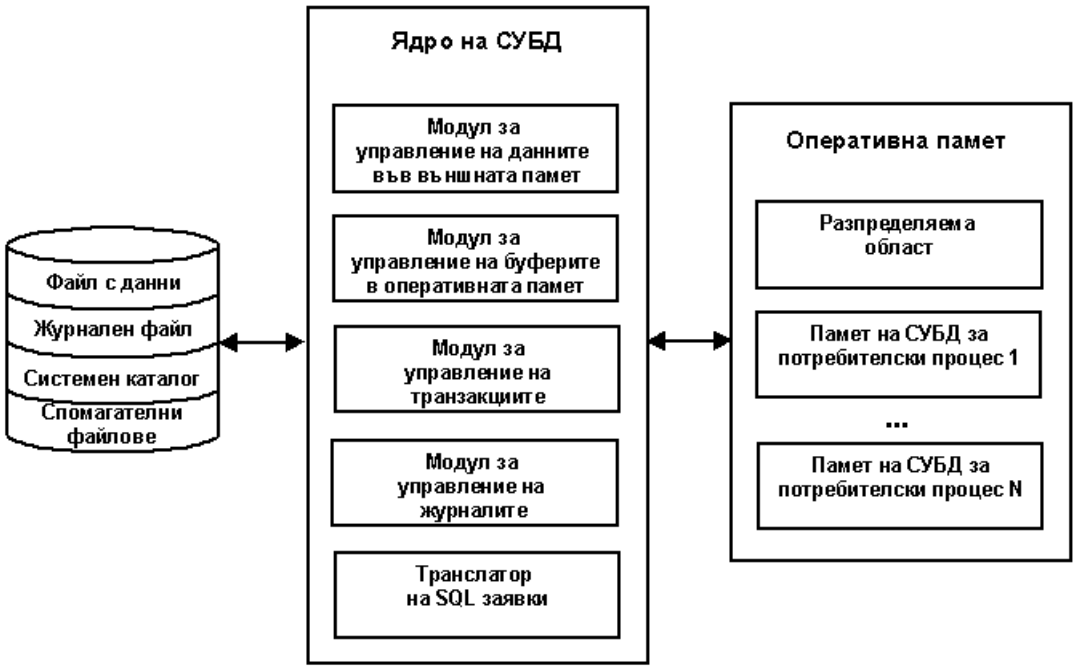
Ако write_TS(X)<=TS(T), тогава се изпълнява read_item(X) от T и read_TS(X) приема стойността на TS(T).

Транзакциите, чието изпълнение е прекратено, получават нови timestamps и се поставят в опашката на чакащите обработка транзакции.

10. Основни функции на СУБД

10.1 Обобщена архитектура на СУБД

Разгледаните до сега различни аспекти на работа на СУБД позволяват да се направят обобщения и да се построи обобщена структура на СУБД. На фиг. 10.1 е показана такава структура.



Фиг. 10.1

Вижда се,че СУБД трябва да управлява външна памет, в която са разположени файловете с данни, журналните файлове и файловете със системния каталог.

СУБД също така управлява оперативната памет, в която се помещават буфери с данни, буфери на журнала, данни от системния каталог, които са необходими за поддържане на целостността на БД, както и за управляване на приоритетите на потребителите. Също така, в оперативната памет, по време на работата на СУБД, се разполага информация, съответстваща на текущото състояние на обработваните заявки, както и стратегиите за изпълнение на текущите заявки.

Модулът за управление на външната памет осигурява създаване на необходимите структури на външната памет за съхраняване на данни, принадлежащи на самата БД, както и такива, използвани за служебни цели. В някои реализации СУБД активно използват възможностите на съществуващите файлови системи, при други те изпълняват всички действия, дори обръщенията към запомнящите устройства. Важно е да се отбележи, че, дори СУБД да използва

файловата система или не, то потребителят не е длъжен да знае как са организирани файловете.

Модулът за управление на буферите на оперативната памет е предназначен за подпомагане на изпълнението на всички останали функции.

Оперативната памет, която се управлява от СУБД, може да се разглежда като съвкупност от буфери, съхраняващи данни, буфери, съхраняващи журнала за транзакциите и област за съвместно ползване. Последната обхваща фрагмент от системния каталог (речник на данните), които трябва постоянно да са в ОП, за ускоряване на обработката, област, в която са разположени компилираните SQL оператори.

Системният каталог в релационните СУБД представлява съвкупност от специални таблици, които се ползват от самата СУБД. Те се създават автоматично. При изпълнение на SQL-заявките СУБД постоянно се обръща към тези таблици. Някои СУБД позволяват ограничен достъп на потребителите до системните таблици, но само за четене. Само системният администратор има право да модифицира някои елементи на тези таблици.

Всяка таблица от системния каталог съдържа информация за структурните елементи на БД. Съществува стандарт за вида и броя на системните таблици, но не е необходимо всяка СУБД да поддържа този стандарт. В различните СУБД системният каталог отразява и някои допълнителни възможности.

Областта SQL, за всеки SQL оператор, съдържа данни за свързването, временни буфери, дървото на съответния израз и стратегията за изпълнението му. Ядрото на СУБД периодично отстранява старите, отдавна неизползвани оператори и освобождава място за нови. След като се направи разбор на конкретен SQL оператор и се изготви стратегия за изпълнението му, то тя се помества в SQL областта и се изпълнява.

Модулът за управление на транзакциите поддържа механизма за фиксиране на транзакции, той е свързан с модулът за управление на буферите в ОП и осигурява съхраняване на информацията, която е необходима след настъпване на аварийно прекъсване. Модулът за управление на транзакциите включва специален механизъм за откриване на състояния на блокировка и реализира една от приетите стратегии за принудително приключване на транзакциите с цел прекратяване на настъпила блокировка.

Важна част от ядрото на СУБД е транслаторът от SQL и блокът за оптимизация на заявките. Оптимизацията на заявките може да бъде разделена на синтактична и семантична.

10.2 Защита на информацията в БД

В съвременните СУБД се поддържа един от двата най-общите подходи за поддържане на безопасност на данните – избирателен и задължителен подход. При всеки от тях като единица от данните, за която се създава система за защита, може да бъде както цялата база, така и всеки обект, който е нейна част.

Двата подхода се различават по следните свойства:

- При избирателното управление конкретният потребител има различни права (привилегии или пълномощия) за работа с обектите на БД. Различни потребители могат да притежават различни права за достъп до един или друг обект. Избирателните права се характеризират със значителна гъвкавост.
- При задължителното управление на всеки обект от данните се присвоява някакво класификационно ниво, а всеки потребител притежава някакво ниво на допуск. По този начин, достъп до даден обект имат само определени потребители със съответстващо ниво на допуск.
- За реализация на избирателния принцип са предвидени следните методи. В БД се въвежда нов тип обекти – самите потребители. На всеки потребител в БД се присвоява уникален идентификатор. За допълнителна защита, освен този идентификатор, всеки потребител разполага и с уникална парола.
- Потребителите могат да бъдат обединени в специални групи. Един потребител може да принадлежи на няколко групи. Дефинира се група PUBLIC, за която се определя минимален стандартен набор от права. По премълчаване се предполага, че всеки нов потребител, ако специално не е указано друго, се отнася към групата PUBLIC.
- Привилегиите или пълномощията на потребителите или групите представят набора от действия, които те могат да изпълняват с обектите на БД.
- Напоследък в съвременните комерсиални СУБД се появява понятието “роля”, което се свързва с поименен набор от пълномощия. Съществуват ред стандартни роли. Предоставена е възможност да се създават нови роли, като в тях се групират различни пълномощия. Въвеждането на роли има за цел да опрости управлението на привилегиите на потребителите. Даден потребител може да притежава една или повече роли.

Най-елементарното ниво концепцията за защита на данните е изключително проста – достатъчно е да се проверяват пълномощията и достоверността.

Проверката на пълномощията се основава на обвързването на всеки потребител или процес в информационната система с набор от действия, които той може да изпълнява с определени обекти на БД. Проверката ан достоверността има за цел да потвърди, че потребителят или процесът, които се опитва да изпълни определено действие, е действително този, за когото се представя.

Системата за дефиниране на пълномощията има йерархичен характер. Системният администратор притежава права от най-високо ниво. Обикновено само той има право да създава други потребители и да определя техните пълномощия. СУБД съхранява информация за потребителите и техните права в системния каталог.

Всеки обект от БД има собственик – това е потребителят, който го е създал. Собственикът на обекта притежава всички пълномощия по отношение на този обект, включително и да предоставя или отнема права на други потребители за работа с този обект.

В SQL стандарта не е определена команда за създаване на потребител, практически всички СУБД предлагат специални процедури за това. На потребителите се предоставя възможност за стартиране на съответна процедура.

В SQL са предвидени два оператора за предоставяне и отмяна на привилегии – GRANT и REVOKE.

Операторът GRANT има следния синтаксис:

GRANT { <списък от действия>| ALL PRIVILEGES}

ON <име на обект> TO {<име на потребител>| PUBLIC}

[WITH GRANT OPTION]

Тук списъкът от действия определя допустимите действия за обект от конкретен тип. Параметърът ALL PRIVILEGES указва, че са разрешени всички действия от предварително определения набор от действия.

<име на обект> - дефинира се името на обекта, за който се отнася операторът.

<име на потребителя> или PUBLIC дефинира на кого се предоставят правата.

Параметърът WITH GRANT OPTION е незадължителен и определя режим, при който не само се дефинират права на даден потребител, но и се разрешава той да предава тези права на други потребители. Той може да предава права само в рамките на позволените му действия.

За отмяна на дадени права се използва оператор REVOKE, който има следния синтаксис:

REVOKE {<списък от операции>| ALL PRIVILEGES}

ON <име на обект>

FROM {<списък от потребители>| PUBLIC | CASCADE | RESTRICT}

Параметрите CASCADE или RESTRICT дефинират по какъв начин се отменят привилегиите. Чрез CASCADE се отменят привилегиите не само на потребителя, който е бил непосредствено споменат в оператора GRANT при предоставяне на привилегиите, но и на всички следващи потребители, на които той е предоставил права съгласно параметъра WITH GRANT OPTION. Чрез RESTRICT се отменят правата само на потребителя, който е непосредствено споменат в оператора REVOKE.

С помощта на един оператор REVOKE е възможна отмяна на права на повече от един потребител, както и на цяла група PUBLIC.

Съществена част от защитата на данните е и проверката на пълномощията. Пълномощията на потребителите се съхраняват в специални системни таблици, а проверката има се осъществява от ядрото на СУБД при изпълнения на всяка операция. Може да се каже, че в БД се построява матрица, в която единият размер съответства на потребителите, а другият – на обектите. Елементите на матрицата съответстват на набора от разрешените операции на конкретен потребител с конкретен обект от БД.

При този модел възникват определени проблеми когато е необходими косвено обръщение към обектите. Те се разрешават чрез организационни методи.

Съществуват и проблеми при работа в мрежа, свързани с проверка на достоверността. Те се явяват елемент на защита на работата в мрежова среда, а не само на защитата на БД. Ето защо компютърна защита на информацията, която се осигурява от операционната система има пряко отношение и към защитата на информацията в БД.