**1. Процесс управления данными, базы данных, системы управления базами данных. Модели данных.**

***Данные*** – это информация, фиксированная в определенной форме для последующей обработки, передачи и хранения.

***Управление данными*** – процесс, связанный с созданием, изменением и удалением данных, организацией их хранения и поиска.

***База данных*** – это совокупность данных, хранящихся вместе и управляемых единым описываемым способом.

***Система управления базой данных*** – это программное обеспечение для использования и модификации БД одним или несколькими лицами.

***Модель данных*** – это совокупность структур данных и операций их обработки.

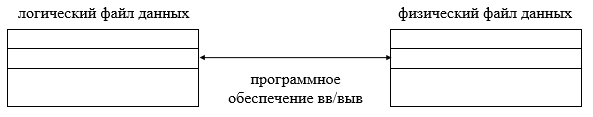
***Иерархическая модель*** – представляет собой совокупность элементов, образующих перевернутое по структуре дерево. Самый верхний уровень иерархии имеет один узел, называемый корнем. Все узлы, кроме корня, связываются с одним и только одним узлом на более высоком уровне иерархии по отношению к ним самим.

***Сетевая модель***, является таковой, если она включает следующие моменты: на входе блока имеются несколько связей 1:М; имеются циклы; наличие петель; наличие связи М:М.

***Реляционная модель*** описывает структуру данных в виде наборов отношений (таблиц). Обязательными объектами данной структуры данных являются: отношения, атрибуты, домены, кортежи, первичные ключи, внешние ключи.

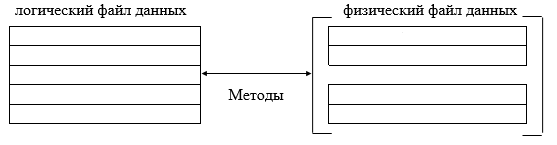
**2. Этапы развития принципов управления данными.**

***1) 60-е годы:***



На этом этапе обработка данных сводилась к вводу/выводу. Данные размещались в файлах последовательно, носителями были магнитные ленты. Любое изменение данных, аппаратных средств приводило к изменению прикладных программ. Логическая структура данных соответствовала физической. Данные использовались только для одного приложения, существовало большое количество копий данных (большая избыточность). Обработка велась в пакетном режиме, не был использован режим реального времени.

***2) Конец 60-х:***



На этом этапе достигнута независимость прикладных программ от аппаратных средств. Существует несложная зависимость между физической и логической структурами данных.

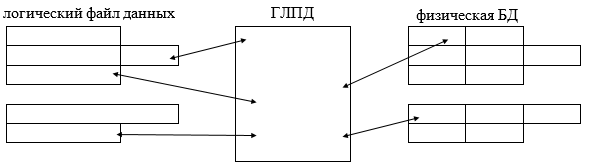
На данном этапе используются следующие методы доступа к наборам данных: прямой; последовательный; индексно-последовательный.

Данные представляются в виде наборов записей. Допускается изменение содержания данных без изменения структуры, это не вызывает изменений прикладных программ. Высокая избыточность. Возможна обработка данных в пакетном режиме и режиме реального времени.

***3) 70-е годы:***

Появление первых СУБД связано с возрастанием объема обрабатываемой информации. Возникла необходимость добавления полей и увеличения размера БД. Поэтому элементами данных стали поля записи. Изменение физической структуры БД не вело к изменению логической структуры (физическая независимость данных). За счет использования одних и тех же данных в различных приложениях была уменьшена избыточность.

***4) 80-90 гг. – 2000-е:***



ГЛПД - глобальное логическое представление данных.

С возрастанием обрабатываемого объема данных стало необходимым изменять и логическую структуру данных. Возникла задача 2-х уровней независимости данных. Логическая независимость данных означает, что общая логическая структура данных может быть изменена без изменения прикладных программ. Физическая независимость данных означает, что физическое расположение данных может изменяться, не вызывая при этом изменений общей логической структуры и прикладных программ.

**3. Требования к организации управления данными.**

Успешное функционирование БД может быть осуществлено только при выполнении ряда требований к ее организации. К таким ***основным требованиям*** можно отнести:

а) контролируемая избыточность

б) низкая стоимость

в) скорость обмена данными

г) простота внесения изменений

д) секретность

з) производительность

**4. Этапы проектирования БД.**

***Цель проектирования***: создать точную и защищенную БД, на основе которой можно гарантировать эффективное построение прикладных программ.



При ***логическом проектировании*** производится анализ ПО и информационных потребностей пользователя. Пользовательские требования выражаются рядом внешних моделей - представлений. Проектирование внешней модели заключается в формализации этих представлений. КМД соответствует общему представлению о БД, то есть она включает представление о структуре данных, их целостности и манипулировании данными. Преобразование ВМД в КМД определяется выбором СУБД. Как внешняя, так и концептуальная модель может быть 3-х видов: сетевая; иерархическая; реляционная.

***Физическое проектирование*** связано с фактической реализацией БД. Оно определяет рациональный выбор структуры хранения данных и методов доступа к ним. Результат физического проектирования - внутренняя модель данных.

**5. Классификация БД (по записям, по моделям, по функциям, по назначению, по организации).**

***По характеру хранимой информации:***

1) Фактографические

2) Документальные

***По моделям:***

1) Иерархические

2) Сетевые

3) Реляционные

***По назначению:***

1) Информационные (хранение)

2) Операционные (с последующей обработкой)

***По способу хранения:***

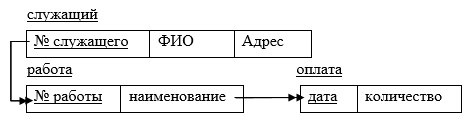
1) Централизованные (БД хранится на одном ПК)

2) Распределенные (БД хранится на многих ПК)

**6. Схемы и подсхемы как способ описания логической структуры данных.**

***Схема*** – описание логической БД. Представляет собой наброски будущих таблиц без включения данных.

Схема состоит из блоков, в которых указываются поля записи, ключевое поле подчеркивается, над блоком ставится имя объекта. Блоки между собой связаны стрелками-связями.



***Подсхема*** – описание части базы данных, которая используется прикладной программой.





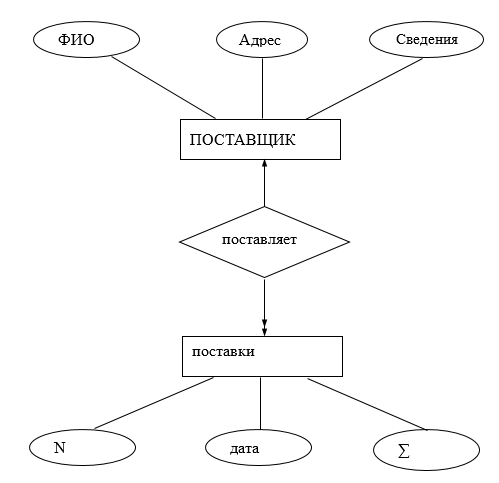
**7. Способы представления исходной информации для построения БД.**

***Неформальная модель*** предметной области, которая используется на этапе инфологического проектирования, является исходной для логического проектирования - это модель типа «объект- связь».

***Назначение этой модели*** – семантическое описание предметной области, а также представление информации для выбора вида моделей и структур данных для следующих этапов проектирования БД.

***Наиболее распространенными являются 3 вида описания неформальной модели***:

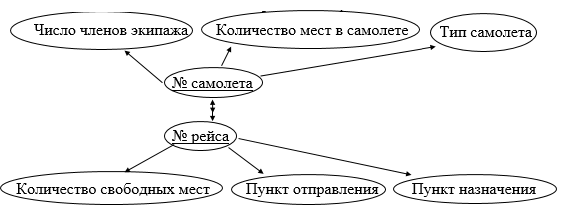
1. *ER - диаграммы*. Прямоугольник обозначает объект, овал – атрибут, ромб – связь.



2. *Диаграммы Бахмана.* Прямоугольник – объект, внутри прямоугольника пишется имя объекта и в круглых скобках могут быть перечислены атрибуты. Объекты связаны между собой направленными ребрами, связи подписываются. Диаграммы предназначены для древовидных, сетевых структур, поэтому всегда существует исходный объект и подчиненный объект.



3. *Овал-диаграммы.* Атрибуты представляются в овалах. Ключевые атрибуты подчеркиваются. Связи представляются в виде направленных ребер между овалами.



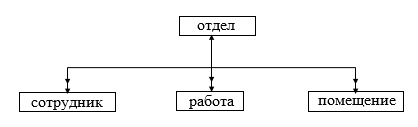
**8. Определения сетевой и древовидной структур данных.**

***Иерархическая*** ***(древовидная) структура*** представляет собой иерархию элементов, называемых узлами. Самый верхний уровень иерархии имеет один узел, называемый корнем. Все узлы, кроме корня, связываются с одним и только одним узлом на более высоком уровне иерархии по отношению к ним самим.

***Древовидные структуры бывают***:

1) однородными (каждый узел представлен одним и тем же типом)

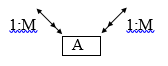
2) неоднородными



Дерево может быть ***переменной*** или ***фиксированной глубины***. Обычно СУБД работает с неоднородными деревьями фиксированной глубины.

***Структура*** называется ***сетевой***, если соблюдаются следующие признаки:

1) когда у 1-го порожденного элемента есть несколько исходных



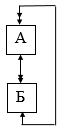
2) когда существует связь М:М (такую сетевую структуру еще называют сложной)



3) наличие петель (цикл, состоящий из 1-го типа записи)



4) имеются циклы (когда предшественник одновременно является и последователем)

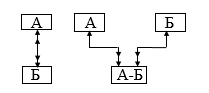


**9. Преобразование сложной сетевой структуры в древовидную структуру данных. Цель преобразования.**

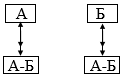
***Цель преобразований*** – перейти к реляционной модели.

Для преобразования ССС в ДСС, сначала необходимо сделать преобразование ССС и ПСС.

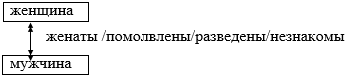
Для этого необходимо сделать развязку отношений М:М путем добавления дополнительной таблицы.



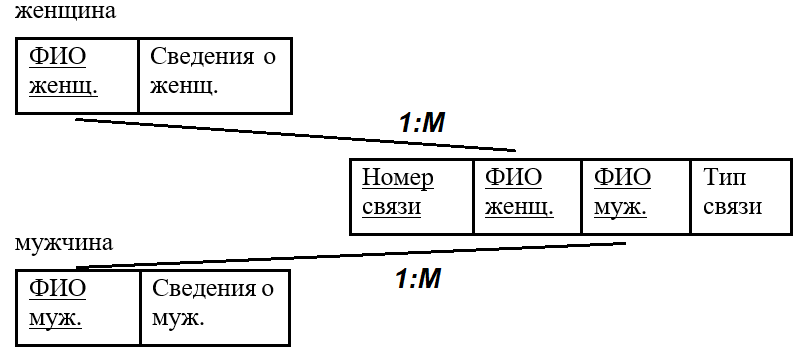
Теперь ПСС можно преобразовать в ДСС введя избыточность.



**10. Множественные отношения.**



Для формализации множественных отношений создают дополнительные отношения связей.



**11. Реляционная модель данных.**

РБД строится на следующих основных принципах:

1) целостность отношений (связь по внешним ключам)

2) нормализация отношений – процесс оптимизации БД

Основой РБД являются плоские таблицы. Обязательными объектами данной структуры данных являются: отношения, атрибуты, домены, кортежи, первичные ключи, внешние ключи.

Процесс приведения древовидной структуры к табличной форме называется нормализацией. Это понятие ввел Кодд.

Задача преобразования древовидной структуры к табличной форме состоит в ведении избыточности. В результате получаем первую нормальную форму (1НФ). Существуют 7НФ. Приведение к каждой из последующих нормальных форм связано с устранением избыточности.

**12. Получение 1НФ из древовидной структуры.**

Чтобы отношение соответствовало 1НФ, необходимо привести это отношение в соответствии 3 условиям:

1) определить первичные ключи

2) устранить повторяющиеся группы

3) привести поля к атомарности

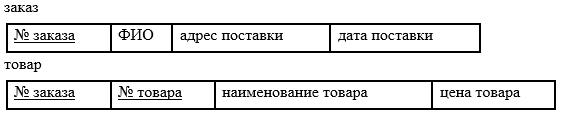
Свойства, которыми должен обладать ключ:

1) однозначная идентификация кортежа

2) отсутствие избыточности (то есть никакой атрибут нельзя удалить из ключа, не нарушая при этом свойства однозначной идентификации)



После приведения к 1НФ:



**13. Сравнение моделей на базе сетевой, иерархической и реляционной структуры.**

***БД классифицируются по объему записей***:

1) малые (около 1к)

2) средние (около 10к)

3) большие (100к)

4) крупные (>100к)

***Для сравнения структур данных существует два критерия***:

1) *Легкость использования*. Основная часть издержек, особенно в малых БД, содержащих порядка нескольких тысяч записей, и средних БД, содержащих порядка нескольких десятков тысяч записей, приходится на написание прикладных программ и формирование запросов.

2) *Эффективность*. Для больших БД издержки приходятся на стоимость памяти и машинного времени, поэтому необходима эффективная реализация.

Исходя из этих критериев, для больших БД используют сетевую или иерархическую структуры, а для малых БД и средних БД - реляционную структуру. Кроме того, критерием решения при выборе структуры является вид запросов. В случае сложных запросов используют реляционную модель.

**14. Целостность данных.**

***Главная задача проектирования реляционной БД*** – понизить избыточность и повысить надежность БД. При этом основное место отводится целостности данных, то есть ограничениям на зависимости между атрибутами отношений.

***Существуют два вида ограничений***:

1) *ограничение, которое зависит от семантики элементов домена* (значение элемента должно находиться в каком-либо диапазоне, либо должны выполняться какие-либо зависимости между различными атрибутами одного кортежа)

2) *ограничение, связанное с равенством значений атрибутов различных отношений* (связь таблиц по внешнему ключу, то есть первичный ключ всегда должен быть определен, а значения внешнего ключа могут быть определены, а могут быть и не определены). ***К таким ограничениям относят*** функциональные зависимости, многозначные зависимости и зависимости соединения.

**15. Определение функциональной зависимости (F-зависимости).**

***Введем следующие обозначения***:

1) A, B, C – обозначения атрибутов

2) X, Y, Z – множество атрибутов

3) a, b, c, x, y, z – значения атрибутов

4) R, S, V – отношения

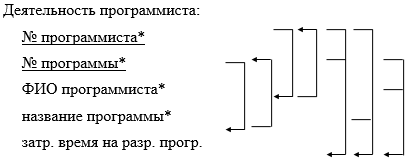
5) U – универсальное отношение (таблица, состоящая из всех доменов и атрибутов БД)

6) XY – объединение множеств атрибутов X и Y

Пусть X и Y – атрибуты отношения R.

Атрибут X функционально ***зависит*** от атрибута Y, если в каждый момент времени каждому значению X соответствует одно и то же значение Y: X→Y

Если имеем запись X#Y, то это означает, что между ними ***нет F – зависимости***.



**16. Определение полной функциональной зависимости.**

Атрибут Y отношения R называется ***полностью зависимым*** от атрибута X, если он полностью зависит от всех атрибутов множества X, но не зависит от какого-либо подмножества Z.

X→Y, Z→Y, где X=Z

**17. Аксиомы функциональных зависимостей.**

Пусть дано: U, множество атрибутов X, Y, Z, W и множество функциональных зависимостей F.

**Аксиома 1** (рефлексивность): если X ⊆ Y ⊆ U то Y → X, такие зависимости называют тривиальными (то есть правая часть содержится в левой: А → АC)

**Аксиома 2** (пополнение): если X → Y и Z ⊂ U, то ZX → ZY

**Аксиома 3** (транзитивность): если X → Y и Y → Z, то X → Z

**Правило 1** (объединение): если X → Y и X → Z, то X → YZ.

**Правило 2** (псевдотранзитивность): если X → Y и WY → Z, то WX → Z

**Правило 3** (декомпозиция): если X → Y и Z ⊆ Y, то X → Z

**18. Вычисление замыкания множества функциональных зависимостей.**

Если есть F = { X→B1, X→B2, ..., X→Bm }, чтобы получить множество F+, необходимо сформировать множество зависимостей вида X→Y, где Y – комбинация атрибутов { B1, B2, ..., Bm }, количество комбинаций два в степени m.

***Алгоритм построения замыкания X+:***

Дано: F, U, X.

Алгоритм сводится к нахождению множеств Х(0), Х(1), ..., Х(n).

1) X(0):=X; i:=0

2) формируем Х(i+1) присоединяя к X(i) атрибуты из правой части функциональной зависимости, в левой части которой стоит множество атрибутов, являющихся подмножеством X(i)

3) i:=i+1; и переходим к пункту 2 до тех пор, пока X(i)#X(i+1)

4) X+:=X(i+1)

***Пример:***

Дано: F = { AB→C, D→EG, C→A, BE→C, BC→D, CG→BD, ADC→B, CE→AG }

X = BD

Найти: X+.

1) X(0):=BD (D→EG)

2) X(1):=BDEG (BE→C)

3) X(2):=BCDEG (CE→AG)

4) X(3):=ABCDEG

5) X+:=ABCDEG

Таким образом исходное сочетание атрибутов БД определяет любую комбинацию из Х+, а, следовательно, Х+ построена и может считаться F+.

**19. Покрытие множеств зависимостей.**

Пусть F и G множества зависимостей универсального отношения (U), F и G – эквиваленты, если совпадают множества F+ и G+.

Проверка эквивалентности F и G: для каждой зависимости X→Y множества F определяют содержится ли эта зависимость в G+. Для этого находим замыкание X+ относительно G+, и, если Y⊆X+, тогда эта зависимость X→Y принадлежит G+. Проверяем принадлежность всех зависимостей из F которые являются подмножеством G+, если все зависимости принадлежат G+, то G покрывает F. Выполняем аналогичную проверку для всех зависимостей G к F+. Если F покрывает G и G покрывает F, то F и G эквивалентны.

***Условия минимальности множества зависимостей F***:

1) все зависимости во множестве F имеют вид X→A, то есть в правой части стоит один атрибут;

2) не для какой зависимости X→A ее удаление из F не приведет к получению эквивалентного множества;

3) не существует такой зависимости X→A, для которой выполняется условие эквивалентности (F-{X→A})∪(Z→A) и F, где Z⊆X.

**20. Операция объединения и проекции схем отношений.**

***Проекция*** – выборка из отношения данных по заданным атрибутам.

SQL: SELECT тип\_самолета, вес\_груза FROM самолет

РА: R = П тип\_самолета, вес\_груза (самолет)



Операция ***объединения*** допустима для отношений, имеющих одинаковые схемы. Если отношение имеют разные схемы, но в схемах присутствуют атрибуты, реализованные на одном домене, то операция Объединение допустима для таких отношений только при приведении этих отношений операцией проекцией к одинаковым схемам.

SQL: SELECT тип\_самолета FROM полет WHERE код\_экипажа = 15 UNION SELECT тип\_самолета FROM самолет WHERE число\_экипажа = 5

РА: R1 = П тип\_самолета σ код\_экипажа = 15 (полет) R2 = П тип\_самолета σ число\_экипажа = 7 (самолет) R=R1UR2



**21. Понятие декомпозиции, свойства декомпозиции.**

***Декомпозиция*** схемы отношений R=(A1, A2, ..., An) представляется множеством отношений ρ=(R1, R2, ... , Rm) таких, что R=R1∪R2∪...∪Rm. Причем R1, R2, ... , Rm необязательно являются непересекающимися.

***Декомпозиция имеет два свойства:***

1) соединение без потерь информации

2) сохранение зависимостей

Не каждая декомпозиция обладает этими двумя свойствами.

Говорят, что декомпозиция ρ обладает свойством соединения без потерь информации, если естественное соединение R=R1∪R2∪...∪Rm дает исходное отношение.

Говорят, что декомпозиция обладает свойством сохранения зависимости, если множество зависимостей F выводимо из схем отношений R.

**22. Проверка свойства декомпозиции соединения без потерь.**

Входные данные: схема отношения R на множестве атрибутов A1, ..., Аn, декомпозиция ρ=(R1, ..., Rn), множество F функциональных зависимостей.

Выходные данные: решение о том, обладает ли декомпозиция ρ свойством соединения без потерь.

Алгоритм:

1) строим таблицу с n столбцами и m строками;

2) столбец j соответствует атрибуту Aj, строка i соответствует схеме отношенийRi, на пересечении j-го столбца и i-й строки помещается символ aj, если атрибут Aj принадлежит схеме отношений Ri, иначе ставится символ bij;

3) рассматриваем полученную таблицу, анализируя зависимость X→Y, ищем строки, для которых совпадают значения по всем столбцам, соответствующим X. При обнаружении таких строк отождествляем их символы в столбцах, соответствующих Y. При этом, если оба символа bij и bil, то делаем их равными либо bij, либо bil; если один из символов равен aj, то все другие символы приравниваем к aj.

4) Повторяем до тех пор, пока не рассмотрим все зависимости из F.

5) Если какая-либо строка будет состоять только из символов aj, то такая декомпозиция ρ обладает искомым свойством, иначе – не обладает.

**23. Декомпозиции, сохраняющие зависимости.**

Формально проекцией F на множество атрибутов Z называется множество зависимостей из X→Y в F+ таких, что XY⊆Z.

Декомпозиция ρ обладает свойством сохранения зависимостей, если из объединения всех зависимостей, принадлежащих проекции F на Ri (i=1...n), логически следуют все зависимости F.

Сохранение зависимостей из F необходимо, так как при нарушении зависимостей:

1. могут обнаружится такие текущие значения Ri, которые не удовлетворяют F, даже в случае, когда ρ обладает свойством соединения без потерь;
2. каждое обновление Ri потребовало бы осуществления соединения для проверки выполнения ограничения целостности для отношения R в соответствии множествам зависимостей F.

Пример: дано R={G, A, I}; G - город; A - адрес; I - индекс;

F={GA→I, I→G}

ρ={R1, R2}, R1={A, I}, R2={I, G}

R1∩R2=I, R2-R1=G, тогда I→G

Декомпозиция обладает свойством соединения без потерь, но не обладает свойством сохранения зависимостей, поскольку, не может быть получена зависимость GA→I.

**24. Определение многозначной зависимости.**

Пусть дано отношение R и множество атрибутов X, Y, которые являются подмножеством R (X, Y ⊆ R).

Говорят, что X ***мультиопределяет*** Y (X →→ Y), если заданному значению X соответствует множество значений Y, при чем Y значения не связаны с множеством атрибутов R-X-Y (Y#R-X-Y).

Если имеется многозначная зависимость, то для нее должны выполняться следующие условия.

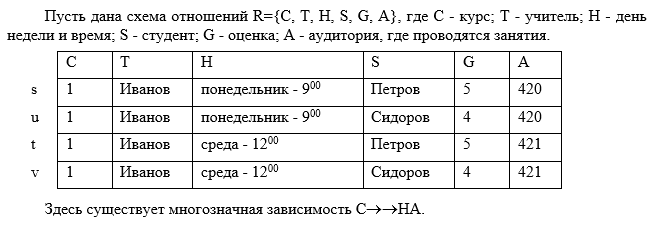
Пусть существует два кортежа s и t в отношении R и X→→Y, и пусть s [X] =t [X] , тогда должно существовать два других кортежа v и u таких, что:

1) u [X] = v [X] = s [X] = t [X]

2) u [Y] = s [Y] u [R-X-Y] = t [R-X-Y]

3) v [Y] = t[Y] v [R-X-Y] = s [R-X-Y]

То есть, если поменять в кортежах s и t местами значения атрибутов множества Y, то получим два кортежа, которые также должны принадлежать отношению R.



**25. Аксиомы многозначных зависимостей.**

**Аксиома 4** (дополнения для многозначных зависимостей): если X→→Y, то X→→U-X-Y

**Аксиома 5** (пополнение для многозначных зависимостей): если X→→Y V⊆W, то XW→→VY .

**Аксиома 6** (транзитивность для многозначных зависимостей): если X→→Y, Y→→Z , то X→→Z-Y

Существуют две аксиомы, связывающие многозначные зависимости с функциональными зависимостями.

**Аксиома 7**: если X → Y, то X→→Y, то есть функциональная зависимость является частным случаем многозначной зависимости.

**Аксиома 8**: если X→→Y , Z⊆Y, W#Y, W→Z, то X→Z .

**26. Замыкание множества функциональных и многозначных зависимостей. Базис зависимостей.**

Задача состоит в следующем: определить следует ли из D зависимость X→→Y либо X→Y. Для этого необходимо построить базис зависимостей для X.

Если Y - X есть объединение каких-либо Yi, то данная зависимость принадлежит D+.

Для вычисления базиса зависимостей X относительно D достаточно найти базис относительно множества многозначных зависимостей M, которое состоит из:

1) из всех многозначных зависимостей D

2) из приведенных функциональных зависимостей X→Z к виду X→→Y1, X→→Y2, ..., X→→Yl, где Yi ∈ { A1, A2,..., An }

**27. Проверка свойства декомпозиции соединения без потерь информации для многозначных зависимостей.**

Данный алгоритм анализа является обобщенным алгоритмом, учитывающим существование как функциональных, так и многозначных зависимостей.

1) Строим таблицу из величин a, b как в алгоритме проверки для функциональных зависимостей.

2) Строим совокупность функциональных таблиц, считая некоторую таблицу Т уже построенной и либо:

а) отождествляем два символа вследствие функциональной зависимости, или

б) выбираем многозначную зависимость X→→Y и две строки t1 и t2 таких, что t1[X] = t2[X], строим новую строку u такую, что

u [X] = t1[X] = t2 [X]

u [Y] = t1 [Y]

u [R - X - Y] = t2 [R - X - Y]

и добавляем ее в таблицу, если ее там еще нет.

3) Так как мы не добавляем никаких новых *a* и *b*, то множество таких таблиц будет конечно. Если какая-либо из таблиц будет иметь строку, содержащую только *а*, то рассмотренная декомпозиция обладает свойством соединения без потерь.

***Существует теорема для декомпозиции двух отношений:*** Пусть R - схема отношения, ρ - декомпозиция на два отношения R1 и R2 , а D - множество функциональных и многозначных зависимостей, то говорят, что декомпозиция ρ обладает свойством соединения без потерь, если:

(R1  ∩ R2 ) →→ (R1 - R2) и (R1  ∩ R2 ) →→ (R2 - R1)

**28. Зависимости соединения. J, ЕJ, ЕМV - зависимости.**

Многозначные зависимости, справедливые для проекции любого вероятностного отношения r для R на подмножестве X, но не имеющего места в самом r называют ***встроенными многозначными зависимостями*** (EMV).

X →→ Y (Z), т.е. X мультиопределяет Y в контексте Z

***Пример:***

R (C, BC, S, Y), где C - курс. BC - вспомогательный курс, S - студент, Y - год.



C →→ BC (Y) или C →→ S (Y)

Многозначные зависимости являются попыткой выделить декомпозицию, обладающую свойством соединения без потерь. В этом случае такая декомпозиция может обладать этим свойством для всех отношений декомпозиций, но не обладать им для некоторого подмножества отношений декомпозиции. Поэтому необходимо определить декомпозиции, которые обладают свойством соединения без потерь, то есть найти так называемые ***J–зависимости***. Нахождение такой декомпозиции приводит к получению 5НФ.

Многозначные зависимости являются частным случаем J–зависимости; J–зависимости являются частным случаем ***EJ–зависимости***, встроенной зависимости соединения.

Отношение R имеет EJ–зависимость, если существует несколько проекций для этого отношения, обладающих J–зависимостями, но в целом отношение не удовлетворяет J–зависимости.

**29. Назначение нормализации и этапы нормализации.**

***Нормализация*** необходима для решения проблемы рационального выбора вариантов схем отношений из возможного множества альтернативных вариантов.

***Схема отношений должна удовлетворять следующим требованиям***:

1) выбранные для отношения первичные ключи должны быть минимальными

2) выбранный состав отношений БД должен быть минимален, то есть отличаться минимальной избыточностью атрибутов

3) при выполнении операций модификации, удаления и включения не должно быть трудностей

4) перестройка набора отношений, приведение новых типов данных должна быть минимальной

5) разброс времени ответа на различные запросы к БД должен быть небольшим

***Выделяют 7 этапов нормализации, соответственно 7 НФ:***



**30. Определение 2НФ и приведение к 2НФ.**

Отношение находится в ***1НФ*** тогда и только тогда, когда каждый атрибут хранит одно единственное значение и не являться ни списком, ни множеством значений (атомарность).

Таблица находится во ***2НФ***, если она удовлетворяет определению первой и все ее поля, не входящие в первичный ключ связаны полной функциональной зависимостью с первичным ключом. Приведение ко 2НФ заключается в необходимости произвести разбиение на несколько таблиц.

***Пример:***

Книги (ID, название, автор, дата рождения автора, издательство, телефон издательства)

*Разбиваем на:*

Книги (ID, название, ID автора, ID издательства)

Автор (ID, автор, дата рождения)

Издательство (ID, издательство, телефон)

**31. Определение 3НФ и приведение к 3НФ.**

Отношение находится в ***3НФ***, если оно находится во 2НФ и никакой не ключевой атрибут функционально не зависит от другого не ключевого атрибута, т.е. нет транзитивных зависимостей (X→Y, Y→Z)=>X→Z

***Пример:***

Сотрудник (ФИО, отдел, телефон)

В отношении существуют следующие функциональные зависимости: ФИО→отдел, отдел→телефон, ФИО→телефон

Зависимость ФИО-телефон является транзитивной, следовательно, отношение не находится в 3НФ.

Приведем к 3НФ:

Сотрудник (ФИО, отдел)

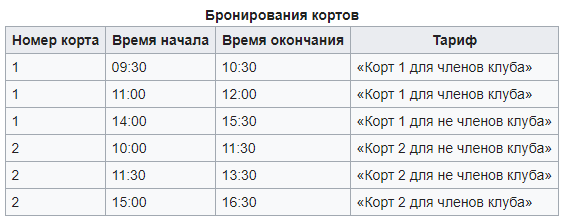
Отдел (отдел, телефон)

**32. Определение НФБК, ДКНФ.**

Отношение находится в ***нормальной форме Бойса-Кодда***, если это отношение находится в третьей нормальной форме и не содержит зависимости атрибутов первичного ключа от не ключевых атрибутов. Иногда нормальную форму Бойса—Кодда называют усиленной третьей нормальной формой, поскольку она во всех отношениях строже по сравнению с ранее определённой 3НФ.

Отношение может находиться в 3НФ, но не в НФБК, при условии, что оно имеет 2 или больше возможных ключа, являющихся составными и имеющих общий атрибут. На практике такая ситуация встречается достаточно редко, для всех прочих отношений 3NF и BCNF эквивалентны.

***Пример:***

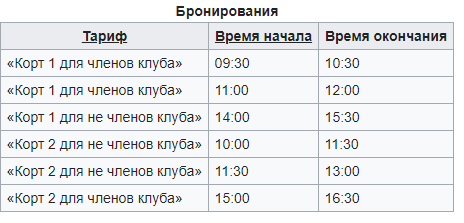


Отношение соответствует 2НФ и 3НФ. Но, существует функциональная зависимость Тариф→Номер корта, в которой левая часть не является потенциальным ключом отношения, то есть отношение не находится в нормальной форме Бойса—Кодда.

Недостатком данной структуры является то, что, например, по ошибке можно приписать тариф «Корт 1 для членов клуба» к бронированию второго корта, хотя он может относиться только к первому корту.

Можно улучшить структуру с помощью декомпозиции отношения на два, получив отношения, удовлетворяющие БКНФ.





Отношение находится в ***доменно-ключевой нормальной форме (ДКНФ)*** тогда и только тогда, когда каждое наложенное на неё определение является логическим следствием ограничений доменов и ограничений ключей, наложенных на данное отношение.

**33. Определение 4НФ и приведение к 4НФ.**

Отношение находится в ***4НФ***, если оно находится в 3НФ и в нём отсутствуют нетривиальные многозначные зависимости.



Так как абитуриент и предмет друг от друга не зависят, то данное отношение содержит многозначную зависимость (Факультет→→Абитуриент|Предмет). Такое отношение обладает целым рядом аномалий. Одна из них состоит в том, что, если мы хотим добавить нового абитуриента на физическом факультете, нам придется добавить столько предметов, сколько находится на физическом факультете и наоборот.

Оптимальным решением проблемы будет декомпозиция отношения на два с атрибутами {Факультет, Абитуриент} и {Факультет, Предмет}.





**34. Нормальные формы: 5НФ, 6НФ, 7НФ.**

Таблица находится в ***5НФ*** тогда и только тогда, когда в каждой её полной декомпозиции все проекции содержат возможный ключ. ***Полная декомпозиция*** – совокупность любого числа проекций, соединение которых совпадает с содержимым исходной таблицы. Таблицы, не имеющая ни одной полной декомпозиции тоже находится в 5НФ.

Переменная отношения находится в ***6НФ*** тогда и только тогда, когда она удовлетворяет всем нетривиальным зависимостям соединения. Из определения следует, что переменная находится в 6НФ тогда и только тогда, когда она не может быть подвергнута дальнейшей декомпозиции без потерь.

**35. Денормализация: цели, задачи.**

***Денормализация*** — намеренное приведение структуры базы данных в состояние, не соответствующее критериям нормализации, обычно проводимое с целью ускорения операций чтения из базы за счет добавления избыточных данных.

В запросах к полностью нормализованной базе нередко приходится соединять до десятка, а то и больше, таблиц. А каждое соединение — операция весьма ресурсоемкая. Как следствие, такие запросы используют ресурсы сервера и выполняются медленно. В такой ситуации следует денормализовать отношения путем сокращения количества таблиц. Лучше объединять в одну несколько таблиц, имеющих небольшой размер, содержащих редко изменяемую информацию, причем информацию, по смыслу тесно связанную между собой.

Также потребляют много ресурсов запросы, в которых производятся какие-то сложные вычисления, тогда имеет смысл добавить в таблицу 1-2 дополнительных столбца, содержащих часто используемые и расчетные данные.

**36. Реляционная алгебра (РА). Основные операции РА.**

***Реляционная алгебра*** – замкнутая система операций над отношениями в реляционной модели данных.

Реляционная алгебра представляет собой набор таких операций над отношениями, что результат каждой из операций также является отношением. Это свойство алгебры называется замкнутостью.

Расширенный начальный вариант представлен Коддом из ***8 операций***. Они делятся на 2 класса:

Теоретико-множественные: пересечение, объединение, разность, декартово произведение

Специальные: проекция, селекция, соединение и деление

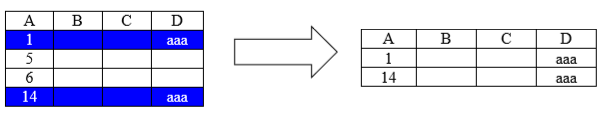
***1. Проекция*** – выборка из отношения данных по заданным атрибутам.



SQL: SELECT B, D FROM name\_table

РА: R = π B, D (name\_table)

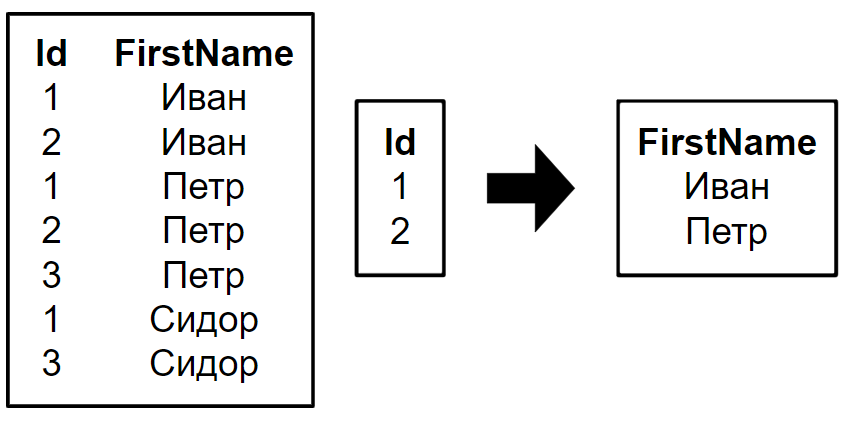
***2.*** ***Селекция*** – выборка из отношения кортежей с ограничением на данные.



SQL: SELECT \* FROM name\_table WHERE D = "aaa"

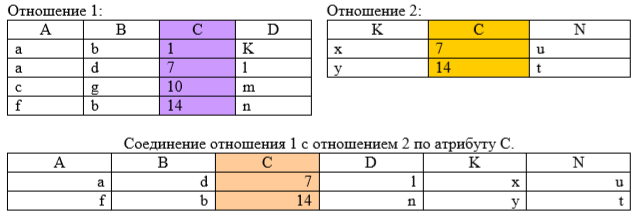
РА: R = σ D = "aaa" (name\_table)

***3. Деление.*** У операции реляционного деления два операнда: бинарное и унарное отношение. Результатом деления двух отношений является отношение, содержащее все значения атрибута первого бинарного отношения, которые соответствуют всем значениям унарного отношения.



РА: R = отношение1 / отношение2

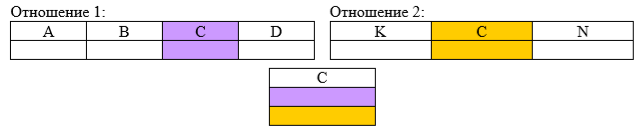
***4. Соединение*** – операцию соединения можно рассматривать, как операцию декартова произведения, при наличии условия ограничения на общий атрибут отношений, по которому реализовано соединение.



SQL: SELECT отношение1.\*, отношение2.\* FROM отношение1, отношение2 WHERE отношение1.C = отношение2.C

РА: R = отношение1 ˃˂ отношение2 C=C

***5. Объединение*** – результатом объединения отношений будут элементы множеств отношения 1 и элементы отношения 2.



SQL: SELECT C FROM отношение1 UNION SELECT C FROM отношение2

РА: R1 = π C (отношение1); R2 = π C (отношение2); R =R1 U R2

***6. Пересечение*** – результатом пересечения отношений будут элементы множеств, принадлежащих одновременно отношению 1 и отношению 2.

SQL: SELECT C FROM отношение1 IN SELECT C FROM отношение2

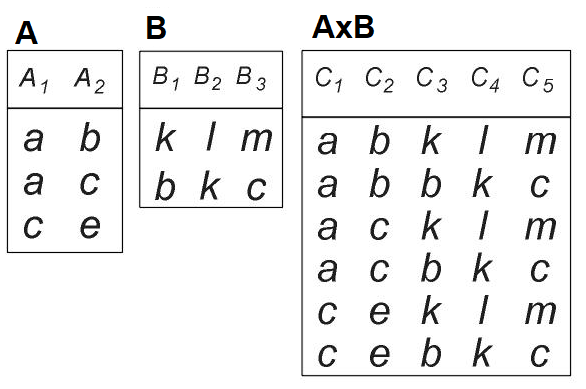
РА: R1 = π C (отношение1); R2 = π C (отношение2); R =R1 ∩ R2

***7. Разность –*** результатом разности отношений будут элементы множеств, принадлежащих отношению 1 и не принадлежащих отношению 2.

SQL: SELECT C FROM отношение1 NOT EXISTS SELECT C FROM отношение2

РА: R1 = π C (отношение1); R2 = π C (отношение2); R =R1 - R2

***8. Произведение*** – результатом будут все возможные комбинации между отношением 1 и отношением 2



SQL: SELECT \* FROM A, B

РА: R = A × B

**37. Дополнительные операторы РА: полусоединение, полувычитание, расширение.**

Операция ***расширения*** порождает новое отношение, похожее на исходное, но отличающееся наличием добавленного атрибута, значения которого получаются путем некоторых скалярных вычислений. Оператор расширения имеет вид:

EXTEND <отношение> АDD <выражение> АS <новый атрибут>

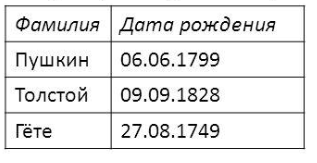
Пример: EXTEND P ADD (Вес\*2.2) AS Фунт

***Полусоединения (SEMIJOIN)*** – содержит только кортежи из отношения R1, такие, которые входят в соединения R1 и R2.

Пример:



Получаем авторов, произведения которых есть в библиотеке:



***Полувычитание (SEMIMINUS)*** – вычитание из исходного отношения R1 полусоединения отношений R1 и R2. Т.е. останутся те кортежи, которые не вошли в полусоединение.

**38. Реляционное исчисление (РИ). Формулы РИ.**

***Реляционную алгебру*** можно рассматривать как высокоуровневый процедурный язык, с помощью которого можно сообщить СУБД, как построить новую таблицу на основе одной или нескольких таблиц, имеющихся в БД. В отличие от этого, ***реляционное исчисление*** представляет собой непроцедурный язык, который позволяет описать, какой будет некоторая таблица, созданная на основе одной или нескольких других таблиц БД. В реляционном исчислении выделяют исчисление доменов и исчисление кортежей.

***Пример БД с отношениями:***

СОТРУДНИКИ (№, имя, зарплата, отдел)

ОТДЕЛЫ (№\_отдела, количество, начальник)

Запрос: выбрать имена сотрудников, являющиеся начальниками отделов с количеством сотрудников более 50.

Средства РА:

1. Соединение отношения СОТРУДНИКИ и ОТДЕЛЫ по условию N, начальник.

2. Ограничить полученное отношение по условию количество больше 50.

3. Проекция результатов предыдущей операции на атрибут имя, N.

В реляционном исчислении (РИ): выдать имена и номера такие, что существует отдел с таким же значением поля начальник и значением количество больше 50.

***Формулы в РИ*** имеют вид {t/ Ψ(t)}, где t - условие, Ψ(t) – формула построенная из атомов и операторов.

***Атомы в РИ бывают:***

1. R(t), где R – имя отношения, t – переменная (кортеж)

2. S[i] Θ U[j], где S, U – два кортежа; i, j – номер атрибутов кортежей; Θ - арифметический оператор сравнения. Θ = {=, <, >, ≤, ≥, ≠}

3. S[i] Θ a, где a – const

***Примеры формул:***

1) Объединение R∪S {t/R(t) ∪S(t)}

2) Разность R – S { t/R(t) S(t)}

**39. Сравнение алгебраических языков и языков исчисления на примере РА и РИ.**

Принято считать, что языки РИ являются языками более высокого уровня, чем языки РА (РА определяет какие операции нужно выполнять, а РИ – предоставляет определить порядок вычислений компилятору). [Остальное в вопросе 38]

**40. Общие стратегии оптимизации запросов. Принципы работы оптимизаторов.**

1.Выполнять операцию селекции по возможности раньше. Это приводит к сокращению промежуточных результатов и как следствие – повышение быстродействия.

2.Целесообразно обрабатывать файлы перед выполнением операции декартова произведения или соединения. Предварительная обработка заключается в сортировке файла и определении индексов с целью ассоциации одинаковых элементом в каждом из файлов.

3.Поиск общих подвыражений выражения. Если результат выполнения какого-либо выражения является небольшим, то можно запомнить, а в дальнейшем используется для вычисления других выражений (не всегда применяется).

4.Сбор в каскады, селекция и проекция. Так как данные операции для 1-го операнда могут выполняться за один просмотр.

5.Комбинация проекции с предшествующими и последующими двухместными операциями.

6.Комбинация селекции с предшествующим декартовым произведениями и выполнение этой операции вместо соединения.

***Функция оптимизатора*** – выбрать наиболее оптимальный (исходя из набора критериев) план выполнения запроса. При формировании оптимального плана, оптимизатор решает следующие задачи

1. Вычисление выражений и операций

2. Преобразование SQL операторов

3. Выбор способа оптимизации – по стоимости или по правилам

4. Выбор путей доступа

5. Выбор порядка соединений таблиц

6. Выбор метода соединений таблиц

7. Определение наиболее эффективного плана выполнения

Оптимизация по правилам (RULE BASED). Подход, при котором учитываются только способы доступа к данным, с зафиксированными приоритетами по эффективности доступа.

Оптимизация по стоимости (COST BASED). Помимо эффективности различных путей доступа к данным, учитывается так же статистика по распределению данных и ресурсов операционной системы.

**42. Эквивалентность и преобразования выражений РА.**

Два выражения в РА являются ***эквивалентными*** (или ), если представляют одно и тоже отображение, то есть при подстановке конкретных значений в выражение E1, E2 получается один и тот же результат.

***Законы эквивалентных преобразований:***

1. Закон коммутативности для соединения и произведения



F – условное соединение.

2. Закон ассоциативности







3. Каскад проекций



4. Каскад селекций



5. Перестановка селекции и проекции



6. Перестановка селекции и декартова произведения



7. Перестановка селекции и объединения



8. Перестановка селекции и разности



9. Перестановка проекции и декартова произведения



10. Перестановка проекции и объединения



**46. SQL. Основные операторы: определение, выборка, обновление.**

Основные SQL команды можно разделить на следующие категории:

1. DDL (Data Definition Language) – язык определения данных.

а) CREATE DATABASE database\_name;

б) CREATE TABLE table\_name ( column1 data\_type(size), column2 data\_type(size), .... );

в) DROP DATABASE database\_name;

г) DROP TABLE database\_name;

д) ALTER TABLE table\_name ADD column\_name column\_type;

е) ALTER TABLE table\_name DROP COLUMN column\_name;

з) ALTER TABLE table\_name MODIFY column\_name column\_type;

2. DML (Data Manipulation Language) – язык манипулирования данными.

а) INSERT INTO table\_name VALUES (value1, value2, value3, …);

б) UPDATE table\_name SET column1 = value1, column2 = value2, ... WHERE condition;

в) DELETE FROM table\_name WHERE condition;

3. DQL (Data Query Language) – язык запросов.

а) SELECT column1, column2 FROM table\_name WHERE condition;

**47. SQL. Агрегатные запросы, коррелированные вложенные подзапросы.**

***Агрегатные запросы:***

Заказы (№ заказа, продавец, стоимость, дата)

1) Получим сумму всех заказов за 2016 год:

SELECT SUM(стоимость) FROM Заказы WHERE дата BETWEEN '2016-01-01' and '2016-12-31';

2) Средняя стоимость заказа:

SELECT AVG(стоимость) FROM Заказы;

3) Минимальная и максимальная стоимость заказа:

SELECT MAX(стоимость), MIN(стоимость) FROM Заказы;

4) Подсчитать количество сделанных заказов:

SELECT COUNT(\*) FROM Заказы;

*Оператор GROUP BY* осуществляет группировку по какому-либо полю:

Пример: вывести сумму всех заказов для каждого продавца по отдельности

SELECT продавец, SUM(стоимость) AS 'Сумма всех заказов' FROM Заказы GROUP BY продавец;

*Оператор HAVING* используется как дополнение к предыдущему. Он необходим для того, чтобы ставить условия для выборки данных при группировке.

Пример: вывести сумму всех заказов для тех продавцов, у которых эта стоимость больше 500

SELECT продавец, SUM(стоимость) FROM Заказы GROUP BY продавец HAVING MAX(стоимость) > 500;

***Подзапросы:***

Поставщики (номер\_поставщика, фамилия, адрес)

Детали (номер\_детали, номер\_поставщика, название)

***Простой подзапрос:***

SELECT фамилия FROM Поставщики WHERE номер\_поставщика IN (SELECT номер\_поставщика FROM Детали WHERE название = "винт");

Простые вложенные подзапросы обрабатываются системой "снизу-вверх". Первым обрабатывается вложенный подзапрос самого нижнего уровня. Множество значений, полученное в результате его выполнения, используется при реализации подзапроса более высокого уровня и т.д.

***Коррелированный подзапрос:***

SELECT фамилия FROM Поставщики WHERE 'винт' IN (SELECT название FROM Детали WHERE номер\_поставщика = Поставщики.номер\_поставщика);

В коррелированном запросе внутренний подзапрос не может быть реализован отдельно: он ссылается на внешний запрос и выполняется последовательно для каждой строки внешнего запроса.

**48. СУБД InterBase. Манипулирование данными. Триггеры, генераторы, представления.**

***InterBase*** – это система управления реляционными базами данных, разработанная компанией Borland.

СУБД построена на ***версионной архитектуре хранения данных***. Этот подход обладает ***рядом преимуществ*** перед блокировочными СУБД:

1. Для восстановления баз данных InterBase после системного сбоя, нет необходимости поддержки лога транзакций.

2. Клиенты, читающие данные, никогда не блокируют клиентов, осуществляющих запись данных.

InterBase обладает целым рядом преимуществ, отличающих его от остальных СУБД:

1) Обновляемые представления

2) Двухфазное подтверждение транзакций

3) Эффективный механизм триггеров

4) Серверная обработка BLOB-полей

5) События

6) Шифрование сетевого трафика, базы данных, файлов бекапа и отдельных столбцов БД

***Язык InterBase SQL*** совместим со стандартом SQL-92. Кроме того, InterBase server поддерживает расширения стандартного подмножества языка SQL за счет добавления собственных функций, написанных в виде библиотеки .dll.

***Генераторы*** используются для генерации уникальных номеров, используемых в ключевых полях. То есть они реализуют те же функциональные возможности, что и поля с автоматическим приращением в других СУБД. Можно указать, чтобы числа генерировались, начиная с заданного начального значения.

Пример создания генератора:

CREATE GENERATOR MyGenerator;

SET GENERATOR MyGenerator TO 1000;

Генераторы не привязываются к какому-либо полю. Они просто позволяют генерировать уникальные числа. Для этого используется функция Gen\_ID(), встроенная в Firebird, которая генерирует целочисленные значения. Она берет генератор в качестве первого параметра и значение шага в качестве второго. Обычно приращение равно 1.

***Триггер*** – это группа операторов языка SQL, которые автоматически выполняется при вставке, модификации или удалении записи.

В СУБД можно создавать триггеры, работающие при следующих шести условиях:

1) до вставки записи (BEFORE INSERT)

2) после вставки записи (AFTER INSERT)

3) до удаления записи (BEFORE DELETE)

4) после удаления записи (AFTER DELETE)

5) до модификации записи (BEFORE UPDATE)

6) после модификации записи (AFTER UPDATE)

CREATE TRIGGER <имя\_триггера> FOR <имя\_таблицы>

[ACTIVE | INACTIVE]

{BEFORE | AFTER} {DELETE | INSERT \ UPDATE}

[POSITION <число>]

AS

[DECLARE [VARIABLE] <переменная тип\_данных>;]

BEGIN

<операторы\_триггера>

END

***Представление*** – это виртуальная таблица, содержимое которой определяется запросом.

Представления дают нам ряд преимуществ. Они упрощают SQL-операции. Они защищают данные, так как представления могут дать доступ к части таблицы, а не ко всей таблице. Представления также позволяют возвращать отформатированные значения из таблиц в нужной и удобной форме.

CREATE VIEW название\_представления [(столбец\_1, столбец\_2, ....)] AS выражение

Пример: CREATE VIEW test\_view (a\_id, b\_id) AS SELECT a.id, b.id FROM a,b;

**49. Язык манипулирования данными QBE.**

По возможностям манипулирования данными при описании запросов языки SQL (Structured Query Language) и QBE (Query By Example) практически эквивалентны. Главное отличие между ними заключается в способе формирования запросов: язык QBE предполагает ручное или визуальное формирование запроса. В то время как использование SQL означает программирование запроса.

Язык QBE позволяет задавать сложные запросы к БД путем заполнения предлагаемой СУБД запросной формы. Такой способ задания запросов обеспечивает высокую наглядность и не требует указания алгоритма выполнения операции – достаточно описать образец ожидаемого результата.

**50. Безмодельные способы управления данными.**

Сравнение SQL и NoSQL:

1) *Структура и тип хранящихся данных*. SQL/реляционные базы данных требуют наличия однозначно определённой структуры хранения данных, а NoSQL базы данных таких ограничений не ставят.

2) *Запросы.* РСУБД реализуют SQL-стандарты, поэтому из них можно получать данные при помощи языка SQL. Каждая NoSQL база данных реализует свой способ работы с данными.

3) *Высокая производительность*. БД NoSQL оптимизированы для конкретных моделей данных, что позволяет достичь более высокой производительности по сравнению с реляционными базами данных.

3) У NoSQL низкая надежность.

4) Доступ к сложным структурам данных NoSQL не поддерживают.

5) NoSQL лучше горизонтально масштабируется.

Категории NoSQL баз данных:

1) БД на основе пар "ключ-значение"

2) Документо-ориентированные БД (MongoDB)

3) Построенные на графах (Neo)

4) Поисковые БД

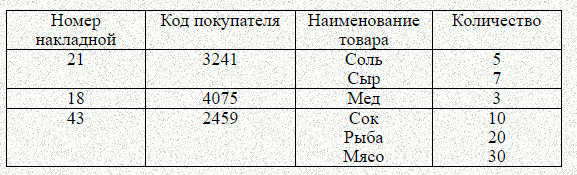
**51. Современные принципы управления данными. Постреляционный подход (UniVers, Dasdb, PostgreSQL, ODMG, РОЕТ и другие).**

Постреляционная модель данных представляет собой расширенную реляционную модель, снимающую ограничение неделимости данных, хранящихся в записях таблиц. Постреляционная модель данных допускает многозначные поля - поля, значения которых состоят из подзначений. Набор значений многозначных полей считается самостоятельной таблицей, встроенной в основную таблицу.

По сравнению с реляционной моделью в постреляционной модели данные хранятся более эффективно, а при обработке не требуется выполнять операцию соединения данных из двух таблиц.

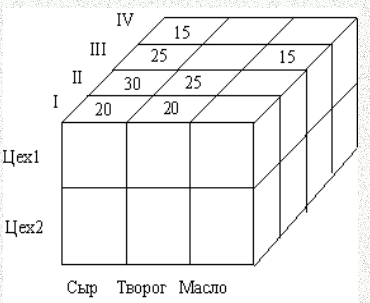
Достоинством постреляционной модели является возможность представления совокупности связанных реляционных таблиц одной постреляционной таблицей. Это обеспечивает высокую наглядность представления информации и повышение эффективности ее обработки.

Недостатком постреляционной модели является сложность решения проблемы обеспечения целостности и непротиворечивости хранимых данных.



Постреляционная модель данных реализована в СУБД UniVers, Bubba и Dasdb.

***Многомерная модель*** представляется в виде многомерных массивов, называемых гиперкубами. В одной базе данных, построенной на многомерной модели может храниться множеств таких кубов, на основе которых можно проводить совместный анализ показателей. Конечный пользователь в качестве внешней модели данных получает для анализа определённые срезы или проекции кубов, представляемые в виде обычных двумерных таблиц или графиков.



POET – объектно-ориентированная СУБД.

PostgreSQL — не только реляционная, но и объектно-ориентированная СУБД.