Оглавление

ТЕОРЕТИЧЕСКАЯ ИНФОРМАТИКА

Свойства информации и алгоритма

Атрибутивные – непрерывность, дискретность, неотрывность информации от физ. носителя, языковая природа информации.

Прагматические – смысл и новизна, полезность, ценность, кумулятивность(небольшой объем, полно отображающий суть), полнота, достоверность, адекватность, доступность, актуальность, объективность и субъективность.

Динамические – рост информации, старение информации.

Свойства алгоритма :

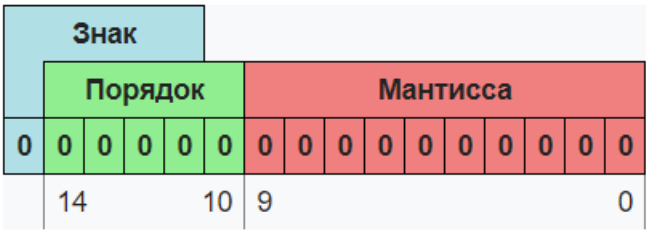
Детерминированность – алгоритм – это некоторое предписание, определяющее пошаговый процесс преобразования входных данных в котором на каждом шаге однозначно определено продолжение или прекращение программы.

Массовость – единообразный подход к преобразованию входных данных в широком множестве этих данных.

Результативность – через конечное число шагов алгоритм выдает какой-либо результат.

Представление вещественных чисел в ЭВМ

Число с плавающей запятой состоит из набора отдельных двоичных разрядов, условно разделенных на так называемые знак, порядок и мантиссу. В наиболее распространённом формате число с плавающей запятой представляется в виде набора битов, часть из которых кодирует собой мантиссу числа, другая часть — показатель степени, и ещё один бит используется для указания знака числа (0 — если число положительное, 1 — если число отрицательное). При этом порядок записывается как целое число в коде со сдвигом, а мантисса — в нормализованном виде, своей дробной частью в двоичной системе счисления. Вот пример такого числа из 16 двоичных разрядов:



Знак — один бит, указывающий знак всего числа с плавающей точкой. Порядок и мантисса — целые числа, которые вместе со знаком дают представление числа с плавающей запятой в следующем виде:

$(-1)^S \times M \times B^E$, где S — знак, B — основание, E — порядок, а M — мантисса.

Нормализованная форма - в которой мантисса десятичного числа принимает значения от 1 (включительно) до 10 (не включительно), а мантисса двоичного числа принимает значения от 1 (включительно) до 2 (не включительно). То есть в мантиссе слева от запятой до применения порядка находится ровно один знак. В такой форме любое число (кроме 0) записывается единственным образом. Ноль же представить таким образом невозможно, поэтому стандарт предусматривает специальную последовательность битов для задания числа 0 и некоторых других полезных значений (Nan, -inf, +inf и т.п.)

Точность:

Число половинной точности (16 бит, 2 байта) – небольшая точность.

Число одинарной точности (32 бит, 4 байта)

Число двойной точности (64 бит, 8 байт)

Число четверной точности (128 бит, 16 байт) - крайне высокая точность.

Перевод из СС в СС

Запишем формулу представления дробного числа в позиционной системе счисления:

$$\mathbf{A}_p = \mathbf{a}_{n-1}\cdot p^{n-1}+\mathbf{a}_{n-2}\cdot p^{n-2} + \ldots + \mathbf{a}_1\cdot p^1+\mathbf{a}_0\cdot p^0 +\mathbf{a}_{-1}\cdot p^{-1}+\mathbf{a}_{-2}\cdot p^{-2} + \ldots + \mathbf{a}_{-m}\cdot p^{-m}$$

В случае десятичной системы счисления получим:

$$\mathbf{24{,}73_2} = \mathbf{2\cdot 10^1+4\cdot 10^0+7\cdot 10^{-1}+3\cdot 10^{-2}}$$

Перевод дробного числа из двоичной системы счисления в десятичную производится по следующей схеме:

$$\mathbf{101101{,}101_2} = \mathbf{1\cdot 2^5+0\cdot 2^4+1\cdot 2^3+1\cdot 2^2+0\cdot 2^1+1\cdot 2^0+1\cdot 2^{-1}+0\cdot 2^{-2}+1\cdot 2^{-3}=45{,}625}$$

Перевод дробного числа из десятичной системы счисления в двоичную осуществляется по следующему алгоритму:

1. Вначале переводится целая часть десятичной дроби в двоичную систему счисления;
2. Затем дробная часть десятичной дроби умножается на основание двоичной системы счисления;
3. В полученном произведении выделяется целая часть, которая принимается в качестве значения первого после запятой разряда числа в двоичной системе счисления;
4. Алгоритм завершается, если дробная часть полученного произведения равна нулю или если достигнута требуемая точность вычислений. В противном случае вычисления продолжаются с предыдущего шага.

Основные операции и законы алгебры логики

Основные операции:

И(AND ^) – логическое умножение, конъюнкция

ИЛИ(OR ^(смотрящая вниз)) – логическое сложение, дизъюнкция

НЕ(NOT ¬) – логическое отрицание, инверсия

Аксиомы алгебры логики:

1. Дизъюнкция двух переменных равна 1, если хотя бы одна из них равна 1.

2. Конъюнкция двух переменных равна 0, если хотя бы одна из них равна 0.

3. Инверсия одного значения переменной совпадает с ее другим значением.

Законы алгебры логики:

1) Законы однопарных элементов:

a. Универсальное множество: X+1=1; X•1=1

b. Нулевое множество: X+0=X; X•0=0

2) Законы отрицания:

a. Двойное отрицание: ^^X=X

b. Дополнительность: X+^X=1; X•^X=0

c. Двойственность (Де Морган): ^(X1•X2)=^X1+^X2; ^(X1+X2)=^X1•^X2

3) Комбинационные законы:

a. Закон тавтологии(идемпотентности): X+X=X; X•X=X

b. Коммутативный закон: X1+X2=X2+X1; X1•X2=X2•X1

c. Ассоциативный закон: X1+(X2+X3)=(X1+X2)+X3; X1•(X2•X3)=(X1•X2) •X3

d. Дистрибутивный закон: (X1•X2)+X3=(X1+X2)•(X2•X3); (X1+X2)•X3 = (X1•X3)+(X2•X3)

f. Абсорбции: X1+X1•X2=X1; X1•(X1+X2)=X1

e. Склеивание: X1•X2+X1•^X2=X1; (X1+X2)(X1+^X2)=X1; ^(X+Y)•(X•^Y)=0

В вопросе нет, но есть в аннотации к вопросам СКНФ и СДНФ:

Совершенной нормальной конъюнктивной формой (СКНФ) функции называется конъюнкция полных совершенных элементарных дизъюнкций.

Совершенной нормальной дизъюнктивной формой (СДНФ) функции называется дизъюнкция полных совершенных элементарных конъюнкций.

Алгоритм построения СДНФ для булевой функции

1. Построить таблицу истинности для функции
2. Найти все наборы аргументов, на которых функция принимает значение 1
3. Выписать простые конъюнкции для каждого из наборов по следующему правилу: если в наборе переменная принимает значение 0, то она входит в конъюнкцию с отрицанием, а иначе без отрицания
4. Объединить все простые конъюнкции с помощью дизъюнкции

Алгоритм построения СКНФ для булевой функции

1. Построить таблицу истинности для функции
2. Найти все наборы аргументов, на которых функция принимает значение 0
3. Выписать простые дизъюнкции для каждого из наборов по следующему правилу: если в наборе переменная принимает значение 1, то она входит в дизъюнкцию с отрицанием, а иначе без отрицания
4. Объединить все простые дизъюнкции с помощью конъюнкции

Примеры:
https://spravochnick.ru/informatika/algebra_logiki_logika_kak_nauka/postroenie_sknf_i_sdnf_po_tablice_istinnosti/

Машина Тьюринга, рекурсия

Под рекурсией понимается метод определения функции через её предыдущие и ранее определенные значения, а так же способ организации вычислений, при котором функция вызывает сама себя с другим аргументом.
f(0)=0
f(n)= f(n-1)+1

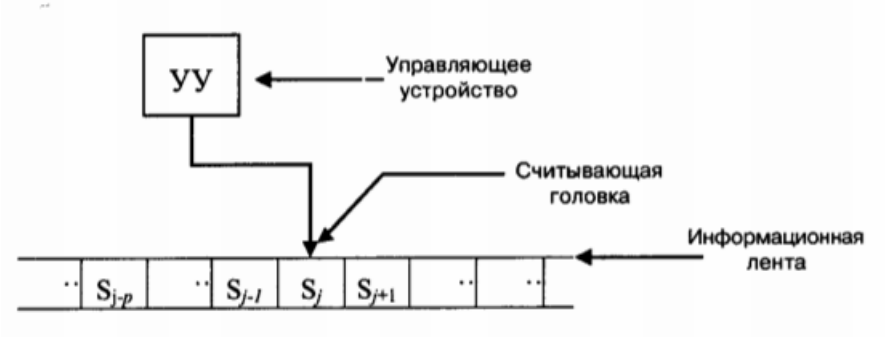
Машина Тьюринга.

Состоит из следующих частей:

1) Информационной ленты, представляющей собой бесконечную(неограниченную) память машины. Лента разделена на ячейки. В каждой ячейке можно поместить лишь один символ, в том числе и ноль.

2) «Считывающей головки» - специальный чувствительный элемент, способный обозревать содержимое ячеек. Вдоль головки информационная лента перемещается в обе стороны так, чтобы каждый рассматриваемый момент времени головка находилась в одной определенной ячейке ленты.

3) Управляющего устройства, которое в каждый рассматриваемый момент находится в некотором «состоянии». Предполагается, что устройство управления машины может находиться в некотором конечном числе состояний. Состояние устройства управления часто называют внутренним состоянием машины. Одно из этих состояний называется заключительным и управляет окончанием работы машины.



Чтобы задать конкретную машину Тьюринга, требуется описать для нее следующие составляющие:

1. Внешний алфавит. Конечное множество (например, A), элементы которого называются буквами (символами). Одна из букв этого алфавита (например, a0) должна представлять собой пустой символ.
2. Внутренний алфавит. Конечное множество состояний головки (автомата). Одно из состояний (например, q1) должно быть начальным (запускающим программу). Еще одно из состояний (q0) должно быть конечным (завершающим программу) – состояние останова.
3. Таблица переходов. Описание поведения автомата (головки) в зависимости от состояния и считанного символа.

Автомат машины Тьюринга в процессе своей работы может выполнять следующие действия:

1. Записывать символ внешнего алфавита в ячейку (в том числе и пустой), заменяя находившийся в ней (в том числе и пустой).
2. Передвигаться на одну ячейку влево или вправо.
3. Менять свое внутреннее состояние.

Одна команда для машины Тьюринга как раз и представляет собой конкретную комбинацию этих трех составляющих: указаний, какой символ записать в ячейку (над которой стоит автомат), куда передвинуться и в какое состояние перейти. Хотя команда может содержать и не все составляющие (например, не менять символ, не передвигаться или не менять внутреннего состояния).

Пример работы : https://inf1.info/turing

Опять же, есть в аннотации, значит желательно знать:

Автомат Мили — конечный автомат, выходная последовательность которого (в отличие от автомата Мура) зависит от состояния автомата и входных сигналов. Это означает, что в графе состояний каждому ребру соответствует некоторое значение (выходной символ). В вершины графа автомата Мили записываются выходящие сигналы, а дугам графа приписывают условие перехода из одного состояния в другое, а также входящие сигналы.

Автомат Мура (абстрактный автомат второго рода) в теории вычислений — конечный автомат, выходное значение сигнала в котором зависит лишь от текущего состояния данного автомата, и не зависит напрямую, в отличие от автомата Мили, от входных значений.

Классификация и эволюция языков программирования

Низкоуровневый язык программирования (язык программирования низкого уровня) — язык программирования, близкий к программированию непосредственно в машинных кодах используемого реального или

виртуального (например, Java, Microsoft .NET) процессора. Для обозначения машинных команд обычно применяется мнемоническое обозначение. Это позволяет запоминать команды не в виде последовательности двоичных нулей и единиц, а в виде осмысленных сокращений слов человеческого языка (обычно английских). Как правило, использует особенности конкретного семейства процессоров. Общеизвестный пример низкоуровнего языка — язык ассемблера.

Высокоуровневый язык программирования — язык программирования, разработанный для быстроты и удобства использования программистом. Основная черта высокоуровневых языков — это абстракция, то есть введение смысловых конструкций, кратко описывающих такие структуры данных и операции над ними, описания которых на машинном коде (или другом низкоуровневом языке программирования) очень длинны и сложны для понимания.

Процедурные (императивные) языки — это языки операторного типа. Описание алгоритма на этом языке имеет вид последовательности операторов. Характерным для процедурного языка является наличие оператора присваивания (Basic, Pascal, C).

Непроцедурные (декларативные) языки — это языки, при использовании которых в программе в явном виде указывается, какими свойствами должен обладать результат, но не говорится, каким способом он должен быть получен. Непроцедурные языки делятся на две группы: функциональные и логические.

Логическое программирование основано на теории и аппарате математической логики с использованием математических принципов резолюций.

В языках функционального программирования основным конструктивным элементом является математическое понятие функции. Функция в математике не может изменить вызывающее её окружение и запомнить результаты своей работы, а только предоставляет результат вычисления функции.

Объектно-ориентированный язык программирования (ОО-язык) — язык, построенный на принципах объектно-ориентированного программирования.

В основе концепции объектно-ориентированного программирования лежит понятие объекта — некой сущности, которая объединяет в себе поля (данные) и методы (выполняемые объектом действия).

Эволюция ЯП.

Первые ЭВМ появились в 1940-х годах и программировались с помощью машинных языков. Машинный код состоял из последовательностей нулей и единиц.

В начале 1950-х годов была осуществлена идея использования символьных имен вместо адресов данных и замены цифровых кодов операций на мнемонические (словесные) обозначения. Язык программирования, реализующий данный подход, получил название Ассемблер.

Дальнейшая эволюция языков программирования привела к появлению языков высокого уровня, что позволило отвлечься от системы команд конкретного типа процессора.

Важное значение для развития высокоуровневых языков программирования имела разработка во второй половине 1950-х годов трех языков – Fortran, COBOL, Lisp. Философия, стоящая за этими языками, заключается в создании высокоуровневой системы обозначений, облегчающей программисту написание программ.

В период 1960-х — 1970-х годов были разработаны основные парадигмы языков программирования, используемые в настоящее время, хотя во многих аспектах этот процесс представлял собой лишь улучшение идей и концепций, заложенных ещё в Fortran, COBOL, Lisp.

Язык Симула, появившийся примерно в это же время, впервые включал поддержку объектно-ориентированного программирования. В середине 1970-х группа специалистов представила язык Smalltalk, который был уже всецело объектно-ориентированным.

В период с 1969 по 1973 годы велась разработка языка Си, популярного и по сей день и ставшего основой для множества последующих языков, например, столь популярных, как C++ и Java.

В 1972 году был создан Пролог — наиболее известный (хотя и не первый, и далеко не единственный) язык логического программирования.

В 1973 году в языке ML была реализована расширенная система полиморфной типизации, положившая начало типизированным языкам функционального программирования.

Каждый из этих языков породил по семейству потомков, и большинство современных языков программирования в конечном счёте основано на одном из них.

БД

Понятие реляционной модели, реляционных объектов данных, целостность реляционных данных, реляционная алгебра и исчисление(на последнее два – задачи)

Реляционная модель - совокупность данных, состоящая из набора двумерных таблиц. В теории множеств таблице соответствует термин отношение, физическим представлением которого является таблица, отсюда и название модели – реляционная.

Реляционная база данных — это набор отношений, имена которых совпадают с именами схем отношений в схеме БД.

Реляционные объекты данных:

Отношение – таблица

Кортеж – строка или запись

Кардинальное число – количество строк

Атрибут – столбец или поле

Степень – количество столбцов

Первичный ключ – уникальный идентификатор

Реляционная алгебра — это теоретический язык операций, которые позволяют создавать на основе одной или нескольких таблиц другую таблицу без изменения исходных таблиц.

Реляционное исчисление представляет собой *непроцедурный язык*, который позволяет описать, *какой* будет некоторая таблица, созданная на основе одной или нескольких других таблиц БД.

Аксиомы Армстронга:

1. Правило **рефлексивности.** Если множество **в** является подмножеством множества A, **ТО A → B.**
2. Правило **дополнения.** Если A → B, то AC → BC.
3. Правило **транзитивности.** Если A → B и B→C, то A → C.
4. Правило **самоопределения.** A → A.
5. Правило **декомпозиции.** Если A → BC, то A → Bи A → C.
6. Правило **объединения.** Если A → B И A → C, то A → BC.
7. Правило **композиции.** Если A → B и C → D, то AC → BD.

Кто хочет подробнее: 443 страница К.Дж.Дейт Введение в системы БД либо гугел

Условные выражения и предикаты SQL

IF условие

{инструкция|BEGIN...END}
[ELSE
{инструкция|BEGIN...END}]

CASE выражение
WHEN значение_1 THEN результат_1
WHEN значение_2 THEN результат_2
.....
WHEN значение_N THEN результат_N
[ELSE альтернативный_результат]
END

Предикаты представляют собой выражения, принимающие истинностное значение. Они могут представлять собой как одно выражение, так и любую комбинацию из неограниченного количества выражений, построенную с помощью булевых операторов **AND**, **OR** или **NOT**. Кроме того, в этих комбинациях может использоваться SQL-оператор **IS**, а также круглые скобки для конкретизации порядка выполнения операций.

Предикат в языке SQL может принимать одно из трех значений **TRUE** (истина), **FALSE** (ложь) или **UNKNOWN** (неизвестно). Исключение составляют следующие предикаты: **NULL** (отсутствие значения), **EXISTS** (существование), **UNIQUE** (уникальность) и **MATCH**(совпадение), которые не могут принимать значение UNKNOWN.

Предикат сравнения представляет собой два выражения, соединяемых оператором сравнения. Имеется шесть традиционных операторов сравнения: =, >, <, >=, <=, <>.

Предикат BETWEEN проверяет, попадают ли значения проверяемого выражения в диапазон, задаваемый пограничными выражениями, соединяемыми **служебным словом AND**. Естественно, как и для предиката сравнения, выражения в предикате BETWEEN должны быть совместимы по типам.

Предикат IN определяет, будет ли значение проверяемого выражения обнаружено в наборе значений, который либо явно определен, либо получен с помощью табличного подзапроса. **Табличный подзапрос** это обычный оператор SELECT, который создает одну или несколько строк для одного столбца, совместимого по типу данных со значением проверяемого выражения. Если целевой объект эквивалентен **хотя бы одному** из указанных в предложении **IN** значений, истинностное значение предиката **IN** будет равно TRUE.

Подробнее: https://metanit.com/sql/sqlserver/1.1.php

1,2,3 нормальные формы, форма Бойса-Кодда

Хочешь понять? Заходи, ниггеры до тебя доебутся
https://habr.com/ru/post/254773/

Нормальная форма — требование, предъявляемое к структуре таблиц в теории реляционных баз данных для устранения из базы избыточных функциональных зависимостей между атрибутами (полями таблиц).

Отношение находится в 1НФ, если все его атрибуты являются простыми, все используемые домены должны содержать только скалярные значения. Не должно быть повторений строк в таблице.

Отношение находится во 2НФ, если оно находится в 1НФ и каждый не ключевой атрибут неприводимо зависит от Первичного Ключа(ПК).

Отношение находится в 3НФ, когда находится во 2НФ и каждый не ключевой атрибут нетранзитивно зависит от первичного ключа. Проще говоря, второе правило требует выносить все не ключевые поля, содержимое которых может относиться к нескольким записям таблицы в отдельные таблицы.

Отношение находится в НФБК, когда каждая нетривиальная и неприводимая слева функциональная зависимость обладает потенциальным ключом в качестве детерминанта.

Многозначные зависимости и четвертая нормальная форма, зависимости соединений и пятая нормальная форма.

В отношении R (A, B, C) существует **многозначная зависимость** R.A -> -> R.B в том и только в том случае, если множество значений B, соответствующее паре значений A и C, зависит только от A и не зависит от C.

Отношение находится в 4НФ, если оно находится в НФБК и все нетривиальные многозначные зависимости фактически являются функциональными зависимостями от ее потенциальных ключей.

Отношения находятся в 5НФ, если оно находится в 4НФ и отсутствуют сложные зависимые соединения между атрибутами.

Типы и свойства транзакций, управление транзакциями

Почитать : https://studfiles.net/preview/5407664/page:26/

Транзакция (Т.) - это неделимая, с точки зрения воздействия на СУБД, последовательность операций манипулирования данными.

Простые транзакции характеризуется 4 классическими свойствами: атомарность; согласованность; изолированность; долговечность (прочность).

Атомарность – Т. должна быть выполнена в целом или не выполнена вовсе.

Согласованность - гарантирует, что по мере выполнения Т., данные переходят из одного согласованного состояния в другое, т.е. Т. не разрушает взаимной согласованности данных.

Изолированность - означает, что конкурирующие за доступ к БД Т. физически обрабатывается последовательно, изолированно друг от друга, но для пользователей это выглядит так, как будто они выполняются параллельно.

Долговечность - если Т. завершена успешно, то те изменения, в данных, которые были ею произведены, не могут быть потеряны ни при каких обстоятельствах.

К категории управление доступом относятся команды для осуществления административных функций, присваивающих или отменяющих право (привилегию) использовать таблицы в БД определенным образом. Каждый пользователь БД имеет определенные права по отношению к объектам БД. Права – это те действия с объектом, которые может выполнять пользователь. Права могут меняться с течением времени: старые могут отменяться, новые – добавляться. Стандартом языка SQL предусмотрены следующие права:

· SELECT – право читать таблицу;

· INSERT – право добавлять данные в таблицу;

· UPDATE – право изменять данные таблицы;

· DELETE – право удалять данные из таблицы;

· REFERENCES – право определять первичный ключ.

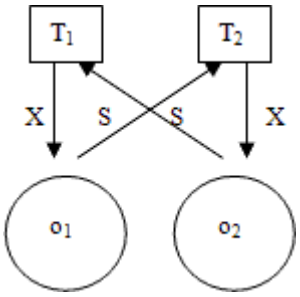
DCL (Data Control Language) – операторы защиты и управления данными.

- Create Assertion – создать ограничение;
- Drop Assertion – удалить ограничение;
- Grant – предоставить привилегии пользователю или приложению для манипулирования данными;
- Revoke – отменить привилегии пользователю или приложению.

Пользователь, создавший таблицу, является ее владельцем. Как владелец, пользователь имеет все права на таблицу и может назначить права для работы с ней другим пользователям. Кроме владельца, права может назначать администратор БД.

Тупиковые ситуации, уровни изоляции, поддержка блокировок в SQL

Офигеть, как в ОС



Ситуация синхронизационного тупика между транзакциями T_1 и T_2

- транзакции T_1 и T_2 устанавливают монопольные блокировки объектов o_1 и o_2 соответственно;
- после этого T_1 требуется совместная блокировка объекта o_2, а T_2 – совместная блокировка объекта o_1;
- ни одно из этих требований блокировки не может быть удовлетворено, следовательно, ни одна из транзакций не может продолжаться; поэтому монопольные блокировки объектов никогда не будут сняты, а требования совместных блокировок не будут удовлетворены.

Под «**уровнем изоляции транзакций**» понимается степень обеспечиваемой внутренними механизмами СУБД (то есть не требующей специального программирования) защиты от всех или некоторых видов вышеперечисленных несогласованности данных, возникающих при параллельном выполнении транзакций. Стандарт SQL-92 определяет шкалу из четырёх уровней изоляции: Read uncommitted, Read committed, Repeatable read, Serializable. Первый из них является самым слабым, последний — самым сильным, каждый последующий включает в себя все предыдущие.

«+» — предотвращает, «-» — не предотвращает.

Уровень изоляции	Фантомное чтение	Неповторяющееся чтение	«Грязное» чтение	Потерянное обновление[3]
SERIALIZABLE	+	+	+	+
REPEATABLE READ	-	+	+	+
READ COMMITTED	-	-	+	+
READ UNCOMMITTED	-	-	-	+
NULL	-	-	-	-

Безопасность в SQL, механизм представлений и подсистема полномочий

В современных СУБД поддерживается один из двух наиболее общих подходов к вопросу обеспечения безопасности данных: избирательный подход и обязательный подход. В обоих подходах единицей данных или «объектом данных», для которых должна быть создана система безопасности, может быть как вся база данных целиком, так и любой объект внутри базы данных.

На самом элементарном уровне концепции обеспечения безопасности баз данных исключительно просты. Необходимо поддерживать два фундаментальных принципа: проверку полномочий и проверку подлинности (аутентификацию).

Проверка полномочий основана на том, что каждому пользователю или процессу информационной системы соответствует набор действий, которые он может выполнять по отношению к определенным объектам. Проверка подлинности означает достоверное подтверждение того, что пользователь или процесс, пытающийся выполнить санкционированное действие, действительно тот, за кого он себя выдает.

SQL поддерживает 3 режима проверки при определении прав пользователя:

1. Стандартный (standard).
2. Интегрированный (integrated security).
3. Смешанный (mixed).

Стандартный режим защиты предполагает, что каждый пользователь должен иметь учетную запись как пользователь домена NT Server. Учетная запись пользователя домена включает имя пользователя и его индивидуальный пароль.

Интегрированный режим предполагает, что для пользователя задается только одна учетная запись в операционной системе, как пользователя домена, а SQL идентифицирует пользователя по его данным в этой учетной записи. В этом случае пользователь задает только одно свое имя и один пароль.

В случае смешанного режима часть пользователей может быть подключена к серверу с использованием стандартного режима, а часть с использованием интегрированного режима.

Декларативная и процедурная поддержка ограничений целостности в бд и отдельно в SQL

Транзакция - это неделимая, с точки зрения воздействия на СУБД, последовательность операций манипулирования данными. Для пользователя транзакция выполняется по принципу "все или ничего", т.е. либо транзакция выполняется целиком и переводит базу данных из одного целостного состояния в другое целостное состояние, либо, если по каким-либо причинам, одно из действий транзакции невыполнимо, или произошло какое-либо нарушение работы системы, база данных возвращается в исходное состояние, которое было до начала транзакции (происходит откат транзакции).

Транзакция обычно начинается автоматически с момента присоединения пользователя к СУБД и продолжается до тех пор, пока не произойдет одно из следующих событий:

- Подана команда COMMIT WORK (зафиксировать транзакцию).
- Подана команда ROLLBACK WORK (откатить транзакцию).
- Произошло отсоединение пользователя от СУБД.
- Произошел сбой системы.

Ограничение целостности — это некоторое утверждение, которое может быть истинным или ложным в зависимости от состояния базы данных. Пример ограничений целостности могут служить следующее утверждение : Возраст сотрудника не может быть меньше 18 и больше 65 лет.

Каждая система обладает своими средствами поддержки ограничений целостности. Различают два способа реализации:

- Декларативная поддержка ограничений целостности.
- Процедурная поддержка ограничений целостности.

Ограничения целостности можно классифицировать несколькими способами:

- По способам реализации.
- По времени проверки.
- По области действия.

Определение 4. ***Декларативная поддержка ограничений целостности*** заключается в определении ограничений средствами языка определения данных (DDL - Data Definition Language).

Определение 5. ***Процедурная поддержка ограничений целостности*** заключается в использовании триггеров и хранимых процедур.

ТИСД

Рекурсия, рекурсивные функции

Под рекурсией понимается метод определения функции через её предыдущие и ранее определенные значения, а также способ организации вычислений, при котором функция вызывает сама себя с другим аргументом.

$$\begin{cases} f(0)=0 \\ f(n)= f(n-1)+1 \end{cases}$$

```
def preorder(tree): //обход дерева в глубину на языке Python-_-
    if tree:
        preorder(tree.getLeftChild())
        preorder(tree.getRightChild())
      print(tree.getRoot())
```

Сравнение алгоритмов

http://bigocheatsheet.com/

Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)

ОС Тенденции развития, типы ОС, ресурсы ВС

Повышение надежности, отказоустойчивости, безопасности ОС, развитие ОС с открытым кодом, развитие виртуализации, сближение ОС мобильных устройств и компьютеров, объединение ОС и сетей. (тупа из башки если не согласны то найдите родителей этих пидарасов)

https://studopedia.org/8-98214.html

Под ресурсом понимают какой-либо объект, который может распределяться внутри вычислительной системы (ВС) между конкурирующими за него процессами. Ресурс выделяется процессу на определенный интервал времени. Ресурсы запрашиваются, используются и освобождаются процессами.

По форме реализации различают:

-аппаратные ресурсы (Hard);

-программные ресурсы (Soft);

- информационные ресурсы.

К аппаратным ресурсам относятся аппаратные средства ВС: процессор, оперативная память, внешняя память, каналы ввода/вывода и периферийные устройства.

К программным ресурсам относятся системные и программные модули, которые могут быть распределены между процессами.

К информационным ресурсам можно отнести переменные, хранящиеся в оперативной памяти, и файлы, хранящиеся во внешней памяти. Примером информационного ресурса являются базы данных.

По способу выделения ресурса различают:

-неделимые ресурсы – предоставляются процессу в полное распоряжение;

-делимые ресурсы – предоставляются процессу в соответствии с запросом на требуемое количество ресурса.

ООП

Разница структурного подхода и ооп

Структу́рное программи́рование — парадигма программирования, в основе которой лежит представление программы в виде иерархической структуры блоков.

В соответствии с парадигмой, любая программа, которая строится без использования оператора goto, состоит из трёх базовых управляющих структур: последовательность, ветвление, цикл; кроме того, используются подпрограммы. При этом разработка программы ведётся пошагово, методом «сверху вниз».

Объе́ктно-ориенти́рованное программи́рование (ООП) — методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования[1].

Идеологически ООП — подход к программированию как к моделированию информационных объектов, решающий на новом уровне основную задачу структурного программирования: структурирование информации с точки зрения управляемости[2], что существенно улучшает управляемость самим процессом моделирования, что, в свою очередь, особенно важно при реализации крупных проектов.

Жизненные циклы

https://studfiles.net/preview/962825/

Жизненный цикл объектов и управление памятью.

Обычно объекты в ООП-программе имеют непредсказуемое время жизни. Это означает, что они создаются и уничтожаются в случайном порядке. Программист не всегда может предсказать, на какой стадии жизненного цикла существуют взаимодействующие объекты. Таким образом, нам нужен механизм управления жизненным циклом объекта и памятью.

В каких случаях надо выделять жизненные циклы (совокупности состояний)?

- создание и уничтожение объекта во время выполнения
- миграция между подклассами
- накопление атрибутов. С изменением значения атрибута меняется поведение объекта (человек в зависимости от возраста изменяет поведение)
- операционный цикл оборудования (лифт, станок)
- объект производится поэтапно (сборка машины на конвеере)
- если возникают объекты с жизненным циклом, и мы хотим реализовать асинхронную схему взаимодействия, то задача или запрос – тоже имеют жизненный цикл.
- динамические связи. Формализуется ассоциативным объектом, и для него выделяется свой жизненный цикл. Объект, являющийся таким «результатом» связи, стоит на более высоком уровне, лучше осведомлен о состоянии системы.
- если для какого-то объекта мы выделили цикл, но он имеет безусловную связь с другим объектом (пассивным), то жизненный цикл нужно выделить и для этого пассивного объекта.

Архитектурный домен, шаблоны для создания прикладных классов

Уууу че нашла

https://github.com/Panda-Lewandowski/Object-Oriented-Programming/wiki/Объектно-ориентированное-проектирование.-Принцип-проектирования.-Архитектурный-домен.-Шаблоны-для-создания-прикладных-классов

Домен – отдельный, реальный, гипотетически и абстрактный мир, населенный отчетливым набором объектов, которые ведут себя в соответствии с предусмотренным доменом правилами. Каждый домен образует отдельное и связное единое целое.

Проверка шаблона на корректность происходит только при его использовании. В зависимости от передаваемых параметров шаблон может работать корректно или некорректно. Однако шаблоны достаточно полезны – если есть несколько функций, совершающих одни и те же действия над разными объектами, можно написать шаблон функции. Шаблон задает функцию для работы с разными типами данных. При создании есть параметр типа.

```
//Пример – определение размера файла.
template <typename/*или class*/ T> unsigned length(FILE *fv)
{
        return filelength(fileno(fv)) / sizeof(T); //возвращает размер в байтах, делённый на количество байт в типе
}
```

```
FILE *f;
...
unsigned Count = length<double>(f);
```

Диаграмма потоков данных действий, процессы и потоки управления, доступ к объектам

ДПДД (Диаграмма потоков данных действий) – обеспечивает графическое представление модулей процесса в пределах действия и взаимодействия между ними. Строится для каждого состояния каждого объекта класса.

На диаграмме каждый процесс рисуется овалом. При написании псевдокода выделяется последовательность действий – здесь мы отходим от этого принципа; процесс может выполняться, когда будут доступны все данные, необходимые для его выполнения.

Правила выполнения для ДПДД:

- процес может выполняться, когда всех входы доступны.
- выводы процесса доступны, когда он завершает своё выполнение.
- данные событий (^ просто стрелка сверху) всегда доступны; данные из архивов данных и терминаторов также всегда доступны

Все процессы можно разбить на четыре типа.

Аксесоры – процессы, которые читают какой-либо атрибут, записывают, создают или уничтожают объекты.

Генераторы событий – стрелочка наружу процесса.

Процессы преобразования – выполняют каки-лбо вычисления.

Процессы проверки – условные переходы.

Концепции информационного моделирования

Концепция

- Выделение сущностей, с которыми мы работаем
- Описание или понятие этой сущности.
- Выделение атрибутов, каждая характеристика которая является общей для всех возможных экземпляров классов выделяется как отдельный атрибут.
- Идентификатор – это множество из одного или множества атрибутов, которое определяет класс.
- Выделение привилегированных атрибутов.
- Выявление связей между сущностями
- Реализация

Для каждой подсистемы:

- Строится описание классов, атрибутов, связей между классами.
- Также строится модель взаимодействия объектов (событийная).
- Модель доступа к объектам.
- Таблица процессов состояний.

Для каждого класса: Выделяем модель состояний (модель переходов состояний).

Для каждого состояния каждой модели состояний: Строится диаграмма потоков данных действий (ДПДД).

Для каждого действия (процесса): Делается описание процесса.

Моделирование ОЙ (×﹏×) ну наверное стоит по Руди еще почитать, если хотите краткие лекции, то в лс пишите мне

https://vk.com/unatart