Серия «Библиотека инженера»

Я. А. Фельдман

Создаем информационные системы

Разработка прикладных информационно-управляющих систем для предприятий, организаций и средней школы — это просто!

с компакт-диском

Москва
СОЛОН-ПРЕСС
2006

УДК 621
ББК 30
Ф 39

Я. А. Фельдман

Создаем информационные системы. — М.: СОЛОН-ПРЕСС, 2006. — 120 с: ил. — (Серия «Библиотека инженера»)

ISBN 5-98003-256-8

В книге подробно описана новая модель данных и ее программная реализация, предложенная и выполненная автором. Цель — дать возможность каждому предприятию построить единую информационную систему высокого качества. Работающая программа **и** общий подход, описанный в книге, дают возможность решить эту задачу. Особе место в книге занимает применение системы в средней школе — как для тестирования учащихся, так **и** для управления всем педагогическим процессом.

К книге прилагается компакт-диск с копиями экранов и демо-версией программы.

КНИГА - ПОЧТОИ

Книги издательства «СОЛОН-ПРЕСС» можно заказать наложенным платежом (оплата **при** получении) по фиксированной цене. Заказ оформляется одним из двух способов:

1. Послать открытку или письмо по адресу: 123242, Москва, а/я 20.
2. Оформить заказ можно на сайте **www.solon-press.ru** в разделе «Книга — почтой».

Бесплатно высылается каталог издательства по почте.

При оформлении заказа следует правильно и полностью указать адрес, по которому должны быть высланы книги, а также фамилию, имя и отчество получателя. Желательно указать дополнительно свой телефон и адрес электронной почты.

Через Интернет вы можете в любое время получить свежий каталог издательства «СОЛОН-ПРЕСС», считав его с адреса **www.solon-press.ru/kat.doc**.

Интернет-магазин размещен на сайте **www.solon-press.ru**.

По вопросам приобретения обращаться: **ООО «АЛЬЯНС-КНИГА КТК»**
Тел: (495) 258-91-94, 258-91-95, сайт **www.abook.ru**
Сайт издательства «СОЛОН-ПРЕСС»: **www.solon-press.ru**
E-mail: **solon-avtor@coba.ru**

ISBN 5-98003-256-8

© Фельдман Я. А., 2006
© Макет и обложка «СОЛОН-ПРЕСС», 2006

Введение

1. Предисловие автора

Что внутри?

Система FTS (Гибкие деревья) помогает управлять предприятием (заводом, НИИ, школой, больницей, мэрией, торговой сетью...)

Система DTS (Динамический толковый словарь) помогает писать сложные тексты (научные исследования, философские трактаты, комплекты технической документации)

Книга + Диск содержат достаточно информации, чтобы установить демо-версию FTS и полную версию DTS. DTS дана вместе с всеми кодами.

Целевая аудитория FTS

1. Руководитель предприятия или зам руководителя предприятия по ИТ. По-новому оценить последние тенденции ИТ — и поставить у себя FTS. (Предприятие = завод, школа, НИИ, поликлиника, муниципалитет, библиотека, ...)
2. Студент, аспирант, научный сотрудник, изучающий ИТ. Возможно, вам придется использовать FTS. Оценки аналогичных систем на фоне данной книги весьма интересны.
3. Администратор FTS, пользователь FTS (учитель, врач, инженер, технолог, ...). Если у вас применяется FTS. Но если описание вам понравилось, попробуйте уговорить ваших руководителей (как в пункте 1)
4. Программист, поддерживающий FTS (SQL, Java) — как в пункте 2.

Целевая аудитория DTS

1. Писатель-философ, писатель-исследователь.

Художественная литература линейна и по замыслу и по исполнению. Однажды дойдя до совершенства, такой автор дол-

жен остановиться. Не то писатель-философ (например, А Ф Лосев) или писатель исследователь (например, А. Т. Фонько) Их книги, по сути, есть попытка линейно и статично изложить систему смыслов устроенную нелинейно (как сеть) и развивающуюся ежедневно и ежечасно, так что книга, выпущенная утром, к вечеру того же дня уже устарела. Установив DTS, такой автор получает возможность использовать готовую программу для динамической упаковки своих материалов в книгу (1) и в сайт (2)

2. Студент, изучающий ИТ

Есть возможность посмотреть, как это сделано: HTML, JSP, Java, JDBC, ANT-скрипт, SQL.

Дополнительное удовольствие для всех перечисленных выше читателей

Средствами DTS изложено описание DTS и описание FTS. (как пример того, что может DTS) — для всех, кого заинтересовали эти системы по причинам изложенным выше

Три параллельных мира

Платон пишет, что есть три параллельных мира: мир *вещей,* мир *идей* и мир *геометрических форм,* сочетающий свойства первых двух миров. Удивительно, но Платон почти угадал: миров действительно три: *материальный, идеальный* и *виртуальный*. Вспомним, что первые СУБД были (1970) иерархическая IMS IBM и сетевая IDBS CODASYL Cullinet. Через некоторое время эти системы отступили под натиском СУБД реляционных (ADABAS (1980) и другие). Такова история вопроса. Однако за *историей* вопроса скрывается его *логика*.

Указанные три модели данных соответствуют указанным трем параллельным мирам. *Виртуальный* мир описывается *реляционной* моделью данных. *Материальный* мир — мир коллективного действия — описывается *иерархической* моделью данных (FTS). Идеальный мир — мир индивидуального мышления — описывается сетевой моделью данных (DTS). Четвертого не дано.

2. Биография автора

Где я учился и работал

Выпишу только те места и те и годы, где и когда я научился чему-то важному (указываю, чему именно)

Одесса
Средняя школа №116 = программирование (1971)

Москва
Московский Государственный Университет им. Ломоносова (Факультет Вычислительной Математики и Кибернетики) = кибернетические модели (1973)
НИИ Зенит (Зеленоград) = АСУ предприятия (1978)
ВЦКП (Зеленоград) = сеть (1982)
НИИ экономики (Минавиапром) = персональный компьютер (1986)

Израиль
Electronic Games = Windows (1993)
Algorithmic Research = C++ (1996)
Ventura Communication Israel = Internet, HTML, Java (1998)

США
Sprint (Kansas City, KS) = NetBeans IDE, Apache Tomcat http-server, JSP, J2EE (2001)

Россия, Карелия
Петрозаводскмаш = СУБД mckoi (2003), Java 5 (2005)

Мои интересы и сферы деятельности

Не зависимо от моей воли мои интересы развиваются сами собой по четырем направлениям. Вот эти направления

1. Поэзия (своя собственная и чужая), поэтический перевод (с английского на русский)
2. Философия, психология, социология, политика, экономика, логика, история науки и искусства, история литературы и театра, история человечества.
3. Практическая педагогика, школа, воспитание и развитие.

4. Программирование, информатика, применение компьютеров, Интернет.

Все эти направления неожиданно переходят одно в другое. Из философии следует, что надо менять школу (абсолютно все российские и многие зарубежные), и ясно как ее надо менять; чтобы менять школу надо хорошо применять компьютеры. Философская жизненная позиция требует поэтической формулы; анализ поэтического текста требует психологических и философских методов. Спрогнозировать успех программного продукта можно с помощью психологии, экономики, философии. Дизайн программного продукта опирается на определенную логику, психологию, философию. Полировка программного кода подобна полировке поэтического текста.

Мои книги

1. Теория уровней и модель человека. М.: Доброе слово: Черная Белка. 2005. — 224 с. — [Соционика и проблемы типологии личности]. ISBN 5-89796-160-3

Автор предлагает оригинальный подход к анализу абсолютно всех форм человеческого поведения и мышления. Книга показывает как применять этот анализ в политике, экономике, социологии, психологии, педагогике, истории человечества, истории науки, искусства, литературы, при анализе литературных произведений, поэтических текстов и поэтических переводов, личных биографий и пр.

2. Педагогика Луны и Педагогика Солнца. — М.:Доброе слово: Черная белка. 2006. — 148 с. — [Теория уровней и модель человека]. ISBN 5-89596 — 186 — 7

Автор предлагает свою педагогическую систему, основанную на идеях первой книги и на анализе важнейших педагогических систем, прежде всего системы Штайнера («вальдорфские школы») и системе Монтессори

Внимание!

Для тех, кто предпочитает сайты, а не книги:

Все понятия этих двух книг (и некоторые новые изложены на диске в виде сайта, сделанного средствами DTS — смотри выше)

3. ЧТО такое FTS?

FTS это сокращение от английского «flexible trees» — «гибкие деревья». Действительно, все происходящее в системе описывается в виде *трех деревьев,* и это действительно *гибкие* деревья — любое из них можно модифицировать «на лету», т. е. без остановки работающей системы (назовем это свойство «абсолютной гибкостью»). Если я когда-нибудь создам свою фирму, она будет называться Feldman Technology Systems, и это еще одна расшифровка сокращения FTS. Возможен симбиоз расшифровок: Feldman Trees — «деревья Фельдмана». Еще один употребительный вариант: Feldman Testing Systems («Тестирующие системы Фельдмана»).

FTS — это, как сказано в заглавии, *конструктор информационных систем.* Он достаточно богат и гибок, чтобы систему можно было наращивать и модифицировать без участия программистов. По моим расчетам, привлечение программистов составляет только 5% от всех случаев развития в FTS, причем из них только 1% приходится на Java-программиста и 4% — на SQL-программиста. Остальные 95% модификаций выполняет администратор, то есть НЕ программист.

Тем, кто покупает готовые системы, а потом оплачивает работы программистов по их привязке и модификации, хорошо известно, что это долгие и дорогие работы. В современной экономике *модифицировать управление* надо непрерывно. Соответственно, модифицировать надо и информационную систему, питающую управление. При этом привлечение программистов (своих или чужих) — вечная головная боль, черная дыра в бюджете предприятия.

Собственно говоря, *идеальная* система должна дать возможность *каждому* предприятию создать *единую* информационную систему *высокого качества.* То есть надо решить три задачи одновременно:

1) доступность;

2) единство;

3) качество.

Отказ от мощного программистского сопровождения (смотри выше) сильно снижает затраты. Это *первая* компонента доступности. Все используемые «стандартные средства» (http-сервер, СУБД, библиотеки) — законно бесплатные, категории Open Source. Это *вторая* компонента доступности. Вы ставите нулевую конфигурацию (один объект, пять типов) и сами наращиваете ее до того вида, до которого вы (и ваши коллеги) *сегодня* созрели. Это *третья* компонента доступности. Все это ставится на стандартный набор — локальная сеть из персональных компьютеров, браузер, Windows XP или 2000. Можно открыть сервер для Интернета и получить доступ из любой точки земного шара без предварительной установки там чего бы то ни было. Это *четвертая* компонента доступности.

Единство достигается постепенным вытеснением «прочих систем», если таковые были. Система прозрачна — это единое информационное пространство. Нашу Вселенную тоже можно считать единым пространством: в этом пространстве нет перегородок. Но FTS-пространство едино в гораздо более сильном смысле. Я бы назвал такое единство «абсолютной связностью», ибо верны следующие свойства:

1. *Перемещение* из любой точки пространства в любую другую точку пространства *происходит быстро* и с ростом пространства время такого перемещения *не растет*.
2. *Поиск* нужной информации *происходит быстро* и наихудшее время такого поиска с ростом пространства *не растет*.

Качество системы тоже складывается из нескольких компонент. Единство пространства, его абсолютная связность — *первая* компонента. Трудно взаимодействуют программы от разных авторов (поставщиков, английское слово «вендор»). Легко взаимодействуют части одной среды. Трудно работать, если время поиска растет с ростом пространства. В нашем пространстве оно не растет.

Гибкость системы, возможность настраивать ее за минуты — *вторая* компонента качества. Но самое главное — система управления доступом, при которой за *каждый* информационный объект отвечает *известно кто,* а если его нет на работе —

известно, кто за него, и один ответственный *найдется всегда*. (Назовем эти правила «абсолютной ответственностью за данные».) Несмотря на кажущуюся очевидность подобных *требований к системе,* никакая другая из известных систем этими качествами не обладает. Это гарантирует качество хранимой информации, и древнее проклятье GIGO (garbage in — garbage out, мусор внутрь — мусор наружу) больше не работает! Такова *третья* компонента качества.

Питер Брукс-младший в книге «Мифический человекомесяц» *(The Mythical Man-Month by Frederick P. Brooks, Jr, 1972)* пишет, что «увеличивать число разработчиков для уменьшения сроков разработки это все равно что тушить пожар керосином». Продолжая его мысль, с которой невозможно не согласиться, в другую сторону, получаем следующее правило: чем меньше разработчиков у системы — тем лучше. А поскольку *меньше одного разработчика не бывает,* окончательно правило звучит так: *самые лучшие системы имеют только одного разработчика*. Речь идет именно о разработчиках, а не о тестерах, которых, разумеется, чем больше, тем лучше. Так вот, предлагаемая вам система имеет *ровно одного разработчика* — и это ее *четвертая* компонента качества, так сказать, абсолютная ответственность за программы.

Итак, реализованные одновременно, *абсолютная* гибкость, *абсолютная* связность и *двойная абсолютная* ответственность (АААА) гарантируют информационной системе *единство, доступность, качество*. Но как достигается это волшебное АААА? Об этом книга.

4. Для кого и как написана эта книга

В классической методологии создания информационных систем (стандарт UML от компании Rational Rose, в 2003 году вошедшей в состав компании IBM) одним из первых этапов проектирования является диаграмма Actor-Use Case, что по-русски лучше перевести как «роли и случаи». Вот пример такой диаграммы:

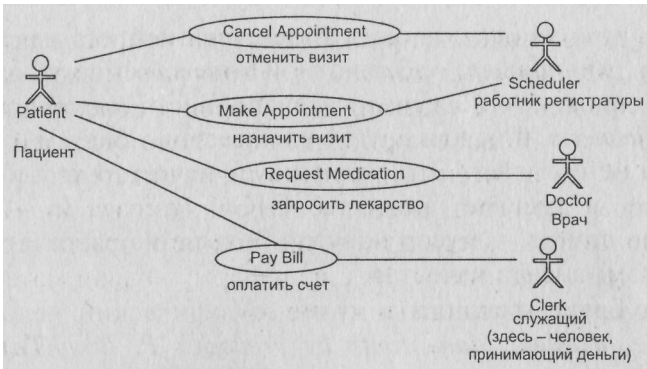

Рис. 1. Диаграмма «Actor-Use Case»

Признавая важность такого анализа не только при создании компьютерных информационных систем, но и при написании книг, выделяю следующие роли своих читателей:

1) Программист:
   a) Архитектор;
   b) Системный программист;
   c) Java-программист;
   d) SQL-программист;
2) Заказчик;
3) Администратор.

Заказчик, Архитектор и Администратор объединены отдельной ролью «Знаток», *Системный программист и Архитектор* объединены отдельной ролью «Специалист», *Программист и Администратор* объединены отдельной ролью «Мастер».

Далее по тексту заголовок второго уровня называет роль, которой адресован очередной фрагмент.

Неотъемлемой частью книги является компакт-диск. На нем есть не только демоверсия системы со всем необходимым для ее установки и проверки, но и сайт с (цветными) копиями экранов, по которым, не устанавливая системы, можно увидеть ее в динамике. Именно поэтому в книге отсутствуют копии экранов — они на диске.

В момент, когда пишутся эти строки, в Москве, в издательстве «Черная белка», (http://izdat.socion.org) выходит моя книга «Теория уровней и модель человека». В данной книге *теория*

уровней будет использована для ответа на вопрос: какую долю рынка *информационных систем предприятия* может захватить FTS при правильном ведении дел? Хорошо обоснованный ответ (минимум 14% всех корпоративных пользователей США и минимум 6% — в России) приятно удивит читателя.

Некоторые понятия, впервые введенные в *указанной* книге, хочу использовать здесь для более полного ответа на вопрос «как написана *эта* книга». Выбирая форму подачи материала, я учитываю следующую «таблицу видов».

Таблица 1

Таблица видов

Как воспринимается информация	Ухо	*Глаз*	*Рука*
Как она хранится	**Текст**	*Картинка*	Схема
Как она применяется	*Образ*	*Сценарий*	Смысл

Ухо мы не используем, но вы *можете* его использовать, читая текст вслух.

Глаз и ***Сценарий*** мы используем и в книге, и при просмотре диска.

Текст, Схема, Смысл — для книги только.

Рука, Картинка, Образ — на диске только.

К вопросу о «чайниках». Умный человек не боится оказаться *чайником,* он знает, что это не навсегда. Я уважаю всех читателей, *включая чайников,* и стараюсь излагать так, чтобы и «чайники», и «специалисты» получали от одних и тех же фрагментов книги большую *пользу* и большое *удовольствие.*

5. Благодарности

Всем, кто мне помогал.
Всем, кто со мною спорил.
Всем, кто меня учил.
Всем, кто писал программы, которые я использовал.

Заказчик

6. Вначале были деньги

Ищу я выход из ворот, но нет его —
есть только вход, и то не тот.
В. С. Высоцкий

Представьте себе, что сегодня утром вы стали хозяином предприятия.

Предприятием я называю группу людей, долговременно и согласованно стремящихся к общему результату. Размер группы, ее юридический статус и профиль работы (школа, завод, поликлиника, торговая сеть) не имеют никакого значения.

Представьте себе, что у вас на предприятии абсолютная власть и что денег у вас немеряно. При этом вы уважаете компьютеры и все, что с этим связано, и очень хотите их применить. Что будете делать?

Во-первых, вы купите компьютеры (10? 20? 150? 380? 1500?), после чего ваши сотрудники зависнут в играх и пасьянсах.

Во-вторых, вы установите сети и оплатите Интернет, после чего одни ваши сотрудники зависнут в почте, другие в чатах и пр. и др.

А дальше? Дальше ситуация будет развиваться по одному из следующих сценариев.

Сценарий первый. Пускаем дело *на самотек.* Каждый из сотрудников начинает сам добывать программы (пиратские копии), сам ставить, сам изучать, сам заводить данные. *Не позднее чем через полгода* взору удивленного наблюдателя открывается живописная картина под условным названием ***«зоопарк программ и данных».*** Надо ли описывать муки какого-нибудь начальника отдела, знающего, что *эти данные точно есть,* но не в силах отыскать, *где именно они есть,* и, наконец, *вводящего* их в систему *еще раз*? Надо ли описывать бесконечные распечатки на бумагу с одного компьютера, сделанные только для того, чтобы кто-то другой набирал эти цифры заново, чтобы ввести их в другой компьютер, стоящий рядом? *Душераздирающее зрелище,* как сказал бы Иа-Иа.

***Сценарий** второй*. Набираем студентов. Для ровного счета человек семь. Назначаем начальника из родственников. Начинаем лепить самодельный софт. Если учесть, что студентов обучали по учебным планам, утвержденным в Москве одиннадцать лет назад, и что учились они не очень, и что все ошибки у них еще впереди, что получим? *Оно, конечно, работает, но пользоваться им невозможно*. А никто и не пользуется. И не надо было включать.

Сценарий третий. Приглашаем серьезных профессионалов. Microsoft, например, или Oracle, или SAP. Платим. Они ставят. Оно работает. Но не удобно. Чтобы было удобно, надо платить еще. И еще. И еще. В конце концов, кончаются деньги. И дальше — смотри выше.

Есть еще ***четвертый** сценарий,* когда покупаются отдельные подсистемы: 1С для бухгалтерии, ПАРУС для производства, ФОКУС-ПОКУС для управления складом и др. (названия и подсистемы выбраны условно). Но системы из коробки просто так не работают — их надо привязывать и настраивать. При этом получается гремучая смесь из предыдущих трех вариантов.

Что же делать? Где выход? Выходом является технология FTS.

Рассмотрим ***минимальный вариант.*** У вас есть *один надежный* человек, способный после некоторой подготовки стать администратором вашей FTS-системы. Предположим, что он не имеет никаких знаний по программированию, но способен переписывать слова латиницей, не теряя при этом букв. Если он хочет и может учиться, мы *за месяц* (двадцать рабочих дней) сделаем из него *нормального* администратора FTS. Упомянутые 5% ситуаций, для которых нужны программисты, может взять на себя служба поддержки.

Вы не обязаны допускать службу поддержки к реальным данным вашей фирмы. Достаточно держать «демобазу», на которой администратор будет демонстрировать службе поддержки свои пожелания и проблемы и на которой он будет проверять новые версии, чтобы самостоятельно перенести их на основную базу. Для этого ему достаточно открыть демобазу в Интернет, и служба поддержки будет работать с ней удаленно. При этом основная база остается закрытой для внешнего мира.

Максимальный вариант. Предположим дополнительно, что у вас есть два трудолюбивых студента, один очень умный, другой не очень. Очень умного обучаем Java, не очень умного SQL. После чего служба поддержки перестает работать над вашей базой (исчезают те самые 5%) и только отвечает студентам на их вопросы.

Между минимальным вариантом (один НЕ программист) и максимальным вариантом (один НЕ программист и два программиста) есть *промежуточные* (один НЕ программист и один программист; один программист, выполняющий работу администратора и пр.)

***Но** в любом случае вам понадобится не более трех человек* с не очень большой зарплатой.

***Внимание!** Надежность администратора должна быть абсолютной. Но это вопрос чести, а честь не купишь ни за какие деньги.*

Нарисованная здесь картина настолько привлекательнее текущих реалий, что напрашивается очевидный вопрос: ***за счет чего возможен такой фантастический выигрыш в эффективности?***

Ведь даже студенты из второго варианта это не два, а семь человек! Что же говорить об остальных вариантах?

Ответ уже дан выше. *Гибкость* системы позволяет *наращивать ее понемногу,* изо дня в день. В то время как *обычно* подсистемы добавляют большими блоками один раз в *три-четыре-шесть* месяцев. Большая толпа программистов нужна для того только, чтобы в пиковую неделю приемки очередной подсистемы успеть наложить все заплатки на все дыры, обнаруженные заказчиком. О качестве такой «подсистемы в заплатках» можно только догадываться. Все же остальное время программистам нечего делать, кроме как висеть в чате или слоняться из угла в угол. Но если вы вздумаете их уволить, то к очередной пиковой неделе вы не соберете нужной толпы.

Вот такая арифметика. Для пущей убедительности приведу диаграмму, на которой показан перечень работ и относительные трудозатраты для некоторой абстрактной подсистемы, по материалам фирмы IBM.

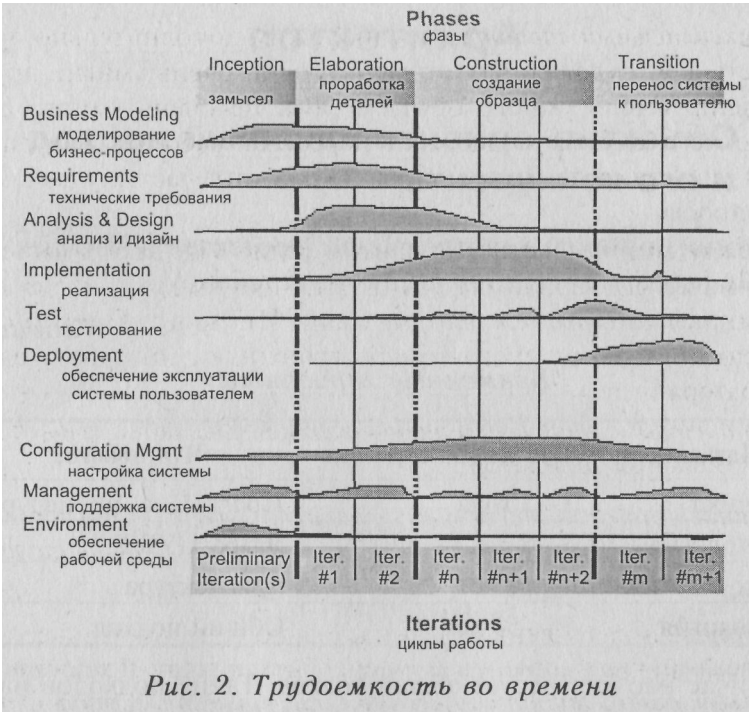

Рис. 2. Трудоемкость во времени

Время, показанное на диаграмме по горизонтали, по мнению IBM, для одной подсистемы составляет от шести до восемнадцати месяцев. У нас единичное добавление занимает от пяти минут до, самое большее (в случае Java-функций, разработанных заново), трех дней.

Архитектор

7. Объектно-ориентированный подход и его конкуренты

Жизнь информационных систем протекает в четырех *временных горизонтах* (смотри следующую таблицу).

Таблица 2

***Временные** горизонты*

Название	Временной горизонт	Что нового?
Техника	1—2 года	Программа, процедура
Технология	3—5 лет	Язык, СУБД
Методология	7—10 лет	Архитектура
Идеология	15—50 лет	Общий подход

Сейчас нас интересует **общий подход.** Таких подходов на сегодня используется несколько, и объектно-ориентированный подход (ООП) — только один из них. Наиболее известные подходы перечислены в следующей таблице.

Таблица 3

Общие подходы

Подход (стандарт)	Единица работы
ООП	Объект
IDEF3	Поток работ
DFD	Поток данных
IDEF0	Функция

Чтобы определить, как соотносятся между собой эти подходы, рассмотрим следующий рисунок (рис. 3).

При создании информационных систем происходит с одной стороны (на рисунке — сверху) осмысление и структуризация

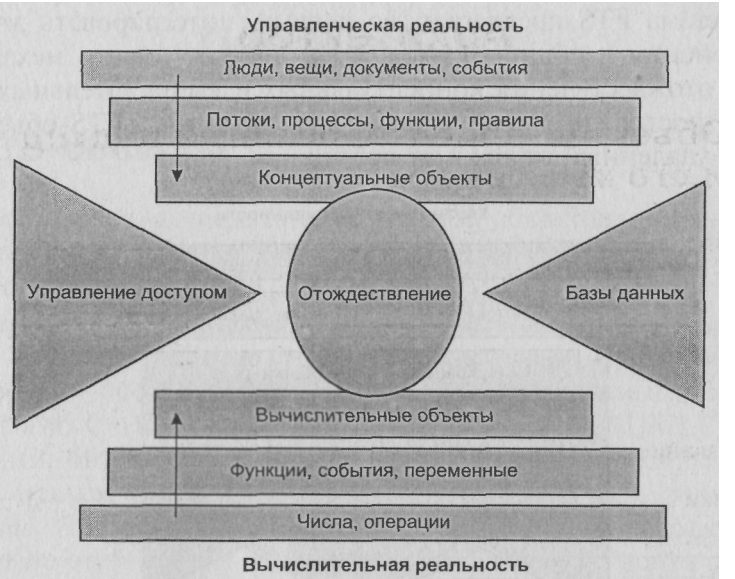

Рис. 3. ООП сегодня

управленческой реальности, с другой стороны (на рисунке — снизу) — обогащение вычислительной среды. Объектно-ориентированный подход доводит дело снизу — до *вычислительных объектов* (объектно-ориентированное программирование; такие языки, как Java и C++), а сверху до *концептуальных объектов* (объектно-ориентированный дизайн, например, с использованием Rational Rose UML). В конце концов, вычислительные объекты строятся как реализация объектов концептуальных, то есть происходит (как показано на рисунке) их фактическое отождествление.

Другие общие подходы просто не доводят структуризацию управленческой реальности до такой глубины. И хотя в том понимании ООП, которое господствует сегодня и которое показано на рис. 3, есть слабые места, замена его на другие подходы из таблицы 2 была бы шагом назад.

Обратите внимание на еще две компоненты, фактически не интегрированные в идеологию ООП, но неизбежно присутствующие в реальной работе. Это Управление доступом и База данных, показанные на рис. 3 двумя треугольниками.

Система FTS предлагает, во-первых, интегрировать эти две компоненты в общий подход и, во-вторых, вместо механического отождествления концептуальных и вычислительных объектов построить некий новый вид объектов (FTS-объекты), равно удаленный от тех и от других.

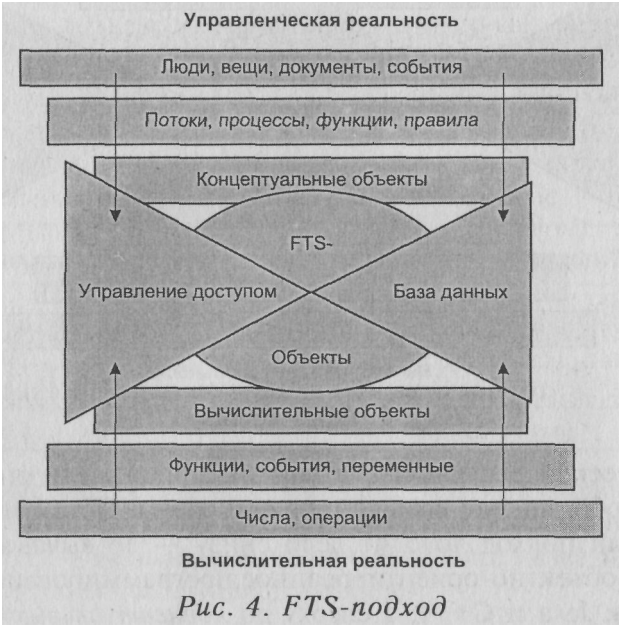

Рис. 4. FTS-подход

8. FTS-объекты

Постараемся вычислить свойства этих объектов, опираясь на здравый смысл и вводя, шаг за шагом, разумные ограничения и предположения.

Предположим, что каждый объект имеет свой ***абсолютный номер*** — целое число от 0 и далее. *Предположим,* что объект с номером 0 — особый. По аналогии с вычислительными объектами (Java, C++) и концептуальными объектами (UML) предположим, что объект содержит элементы данных (member variables), причем у каждого элемента данных есть *имя, формат* и *значение.* Предположим, что объекты как-то между собою связаны и по этим связям от начальной точки можно дойти до искомого объекта. Далее возникает простой вопрос: как хранить объекты?

Теоретически возможны два ответа на этот вопрос:

1) файловая система;

2) СУБД.

Таблица 4

СУБД и файлы: сопоставление

	Поиск по связям	**Набор форматов**
СУБД	Быстрый	Ограниченный
Файлы	Медленный	Неограниченный

Исходя из сказанного, наилучшим решением представляется следующая комбинация двух подходов.

1. Объект делится на управляющие элементы и все остальные.
2. Связи устанавливаются только с использованием управляющих элементов. Здесь список форматов может быть ограничен, но необходим быстрый поиск. Поэтому управ-

ляющие элементы хранятся в СУБД, один элемент — одно поле.

3. Остальные элементы, форматы которых попали в список форматов СУБД, хранятся в СУБД (это работает быстрее и занимает меньше места на диске). Один элемент — одно поле.
4. Элементы, форматы которых не попали в список форматов СУБД, хранятся в файлах, один элемент — один файл. Соотнести файл с объектом, элементом которого он является, позволяют специальные правила вычисления и анализа путей.

Специалист

9. Архитектура и базовые средства

Будем исходить из того, что в работе система должна быть максимально проста и надежна, дешева и законна, легка в освоении и управлении. Поэтому остановимся на следующих решениях.

Один из компьютеров будет сервером. Он будет под особым контролем, все вычисления и преобразования, поиск данных будут происходить там. Остальные компьютеры будут использоваться только для отображения и ввода информации. То есть *остановимся на архитектуре «клиент-сервер»*.

Предположим, что связь идет по протоколу HTTP, а на клиентских машинах установлен *браузер MS IE 5.5 или выше*. Это позволяет нам считать клиентской машиной любой (не очень старый) компьютер, имеющий доступ к серверу по сети. Никаких специальных (предустановленных) клиентов не нужно. Если открыть сервер для Интернета, то вы сможете работать из дома, из другого города и пр.

Браузер MS IE входит в стандартный комплект поставки MS Windows, MS Windows XP является стандартной операционной системой персональных компьютеров, а персональный компьютер сегодня есть везде. Поэтому в качестве машины-сервера выбираем персональный компьютер с операционной системой *MS Windows XP или MS Windows 2000* (самая надежная из предыдущих версий).

С некоторых пор в Интернете можно найти (и загрузить на свой компьютер) программы для бесплатного использования. Обычно такие программы размещаются вместе с исходным кодом. Такие программы называются Open Source. Эти программы привлекают программистов со всего мира, и существуют форумы, где эти программисты высказывают свои мнения о достоинствах и недостатках этих программ. Эти мнения гораздо более объективны, чем любая фирменная реклама, и служат достаточным основанием для оценки качества конкретного продукта. Опираясь на эти мнения и руководствуясь собствен-

ной интуицией и опытом, я выбрал в качестве базовых следующие Open Source-программы.

Язык программирования Java. Основной сайт:

http://java.sun.com.

Базовые библиотеки — J2SE (на сегодня последняя версия 5.0) и J2EE (на сегодня последняя версия 1.4.2). Тем, кто программирует на Java, рекомендую среду разработки NetBeans IDE, на сегодня последняя версия 4.1. Можно загрузить с того же сайта или сайта http://www.netbeans.org.

Для того чтобы *машина-сервер* могла отвечать на запросы с клиентских машин (из браузера, по протоколу HTTP), на ней надо установить программу *http-сервер*. Open Source-проект Jakarta, расположенный по адресу http://jakarta.apache.org, содержит http-сервер Apache Tomcat (на сегодня рекомендуемая версия — 5.5.9) по адресу http://jakarta.apache.org/tomcat/index.html. Из программ этого проекта нам пригодится библиотека для загрузки файлов с клиента на сервер (File Upload), доступная по адресу http://jakarta.apache.org/commons/fileupload/.

В качестве СУБД я использую DBMS Mckoi, сайт

http://mckoi.com.

Вот и все. Все перечисленные программы сделаны в технологии Java по одним и тем же стандартам и взаимодействуют друг с другом без проблем.

Программист

10. Абсолютные номера

Эвристический принцип простейших решений. (Эврика — «открыл», греч. Эвристики — приемы поиска решений.) Проектируя нечто, думай о задачах, которые возникнут вокруг этого нечто. Закладывай при проектировании такие свойства, чтобы эти задачи решались как можно проще.

Мы уже применяли этот принцип, когда обсуждали, где будут храниться данные — под СУБД или в файловой системе. При этом задачи (ускорить поиск, сэкономить место на диске, не ограничивать список форматов) прямо повлияли на выбор свойств проектируемой модели.

Для простоты предположим (пока), что все данные хранятся только под СУБД и что это реляционная СУБД. Для простоты предположим, что в одной записи содержатся записи только одного объекта (но не двух или более). При этом допустимо хранить данные одного объекта в записях разных таблиц. (Хранение данных одного объекта в разных записях одной таблицы недопустимо, этот случай мы рассмотрим позже.)

Как нам собрать воедино все записи, относящиеся к одному объекту? Для этого предположим следующее.

1. Система генерирует абсолютные номера (0, 1, 2, ...). Верхняя граница номеров так велика, что мы можем считать ее бесконечной.

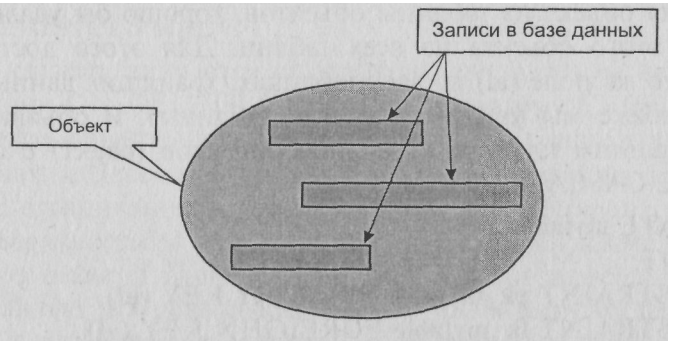

Рис. 5. Записи в базе данных

2. Каждый объект имеет номер (id) из этого ряда. Разные объекты имеют разные номера.
3. В каждой записи, содержащей данные объекта, есть такое поле (id) и его значение равно номеру объекта.

Смотри следующий рисунок.

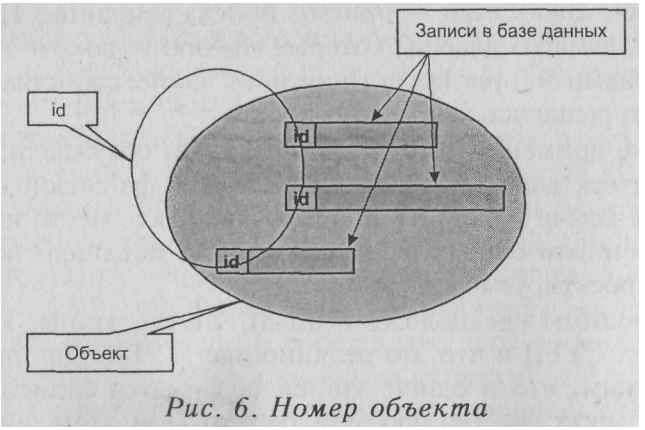

Рис. 6. Номер объекта

Предположим, что нам надо перечислить все объекты, известные системе. Для этого *хорошо бы иметь «таблицу объектов»*. При создании всякий объект должен попадать в эту таблицу один и только один раз. При уничтожении любого объекта строчка в этой таблице, соответствующая этому объекту, должна удаляться. Чтобы быстро искать объект в этой таблице, хорошо бы иметь там указанное поле (id) и объявить его первичным ключом (PRIMARY KEY).

Удаляя объект из таблицы объектов, хорошо бы удалить все данные этого объекта из всех таблиц. Для этого достаточно иметь это же поле (id) во всех таблицах, хранящих данные объектов (позже мы увидим и другие таблицы), и объявить эти поля внешним ключом к таблице объектов **(object)** с опцией DELETE CASCADE, вот так:

```
CREATE mytable
(id INT, ...
CONSTRAINT pk_mytable PRIMARY KEY (id),
CONSTRAINT fk_mytable FOREIGHN KEY (id)
REFERENCES object ON DELETE CASCADE,...)
```

Объявив (id) первичным ключом таблицы данных, мы ускоряем процесс поиска данных. После такого объявления одному объекту не может соответствовать в таблице данных более одной записи (см. выше).

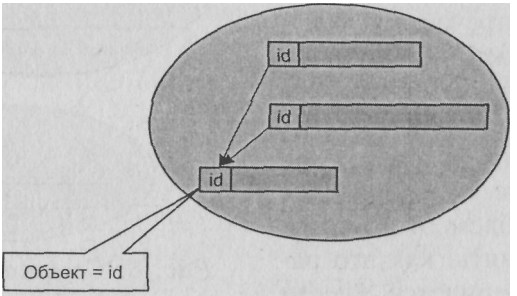

Рис. 7. Связь по ID

11. Ключи

Часто встречается такая ситуация, когда некоторые поля одних объектов указывают на другие объекты. Для реализации под СУБД данного механизма лучше всего объявить некоторые поля ***ключами*** (наряду с *иными форматами* данных под СУБД). *Это поле всегда будет целочисленным.* «Ссылку на объект» реализуем следующим образом (рис. 8).

```
CREATE mytable(.... , mykey INT DEFAULT 0, ...
CONSTRAINT fk_mytable_mykey FOREIGN KEY (mykey)
REFERENCES object
```

Этот случай отличается от предыдущего, здесь не использована опция ON DELETE CASCADE, и поэтому СУБД не даст уничтожить объект, на который есть хотя бы одна ссылка.

Мы накладываем на нашу базу данных следующие дополнительные ограничения: *никакая таблица данных (кроме **object,** о ней поговорим особо) не имеет никаких внешних ключей, кроме описанных выше.* В том числе в нашей модели запрещены ключи составные (из нескольких полей), ключи других форматов (строки символов и пр.), ключи к иным полям таблиц кроме поля id таблицы **object.**

Наличие значения по умолчанию 0 для ключей очень полезно. Почему оно полезно, мы объясним позднее.

В управленческой реальности часто встречаются *документы,* содержащие *«значение из справочника»,* например «сотрудника», «материал» и пр. Использование ключей, как описано выше, — наилучшее решение проблем этого рода. Но чтобы понять, как это работает, нам придется ввести еще несколько понятий.

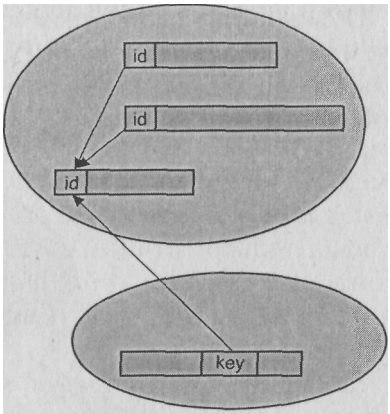

Рис. 8. Ссылка на другой объект

12. Объекты и типы

Предположим, нам надо для данного объекта определить все его данные. Эта задача имеет несколько решений.

Первое решение. Дан номер объекта (X). Переберем все таблицы данных и выберем все строчки, для которых **id=X.** Недостаток этого варианта очевиден: с ростом системы число таблиц растет и, соответственно, при таком подходе будет расти время вывода любого элемента.

Второе решение. Для каждого объекта известно, в *каких именно таблицах* лежат его данные. Эту информацию можно записать в таблицу object, раз уж для каждого объекта в этой таблице есть строчка.

Объекты, для которых это «*в каких именно таблицах*» одинаково, будем называть *«однотипными»* или *объектами одного и того же типа.*

Предположим, нам необходимо перечислить все типы всех объектов системы. Для этого хорошо бы иметь отдельную таблицу ***type.*** Предположим, что все типы имеют абсолютные номера (0, 1, 2, ...). Предположим, что эти номера генерируются системой. Предположим, что генератор номеров объектов и ге-

нератор номеров типов работают независимо. (Простейший вариант реализации таких генераторов в СУБД — SEQUENCE.)

Простейший способ указать таблицу, в которой хранятся данные объектов этого типа — поле **(code),** в котором хранится имя таблицы. Значение этого поля можно считать коротким именем типа (кодом). *Код типа подчиняется всем ограничениям, которые накладывает СУБД на имя таблиц.* (Буквы и цифры только; только латиница; первый знак — буква; подчеркивание считается буквой.) Мы вправе ввести свои *дополнительные ограничения* на это поле. (Буквы только маленькие; строки в кавычках запрещены.)

Кроме короткого имени выгодно иметь длинное имя типа **(name).** Длинное имя может содержать любые знаки (кроме одинарной кавычки, почему — объясню позже). Здесь возможны русские имена любой длины, заглавные буква и пр.

Внимание! *Во всех строках система заменит несколько пробелов подряд на один пробел, а пробелы в начале и в конце строки отсечет.*

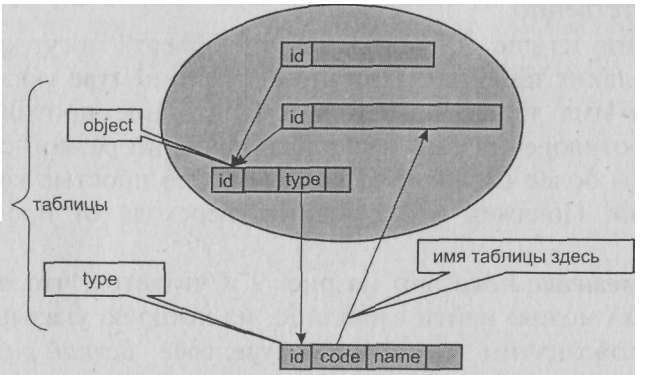

Рис. 9. Таблица типов

Нам выгодно объявить поле **type** таблицы **object,** указывающее на таблицу **type,** — внешним ключом, вот так:

```
CREATE TABLE type (id INT,
       code TEXT NOT NULL,
       name TEXT DEFAULT '',...
```

```
CONSTRAINT pk_type PRIMARY KEY (id),
CONSTRAINT uq_type UNIQUE (code),
CONSTRAINT fk_type_id2 FOREIGN KEY (id2)
 REFERENCES type on delete cascade,
CONSTRAINT ch_type_code CHECK
(code REGEX '[A-z_]+[A-z_0-9]*')
```

Как и в случае с ключами (см. предыдущий пункт), СУБД не даст удалить строчку из таблицы **type,** если в системе есть по крайней мере один объект этого типа.

Другие ограничение (см. выше) проверяют средствами СУБД уникальность значения в поле **code** (UNIIQUE) и ограничения, указанные выше (CHECK) и описанные *методом регулярных выражений* (REGEX).

13. Наследование и сборка

По аналогии с таблицей type в таблице object мы также введем поля **code** и **name** для короткого и длинного имени объектов соответственно.

Как видно из рис. 9, данные одного объекта могут храниться в нескольких таблицах. При этом в таблице **type** указано только одно имя таблицы. Это — объективное противоречие. У этого противоречия есть несколько решений разной сложности, причем более сложные включают более простые как частный случай. Покажем эти решения, переходя от простого к сложному.

Первое решение показано на рис. 9. Считается, что *все* данные объекта можно найти в таблице, на которую указывает *код типа.* Мы обозначим это поле как **type::code.** *Всякий раз,* когда вы встретите подобное выражение, понимайте его как *имя таблицы и имя поля,* разделенные двойным двоеточием. Кстати сказать, ниже мы еще увидим ситуации, в которых первое решение — лучшее из решений.

Второе решение. Таблица **object** содержит строчку для каждого объекта. Поэтому для данного объекта надо взять данные из таблицы **object** и из таблицы с именем, которое содержится в поле **type::code.** Допустим случай, когда это не две таблицы, а

одна. Для простоты, *предположим,* что тип, у которого **type::code= 'object'**, имеет **type::id=0,** то есть это *«нулевой тип»*.

Третье решение. Вспомним, что в языке программирования Java есть «наследование классов». При этом верны следующие правила.

1. Всякий Java-класс, кроме класса **Object,** унаследован от одного из классов, известных системе.
2. Если продлить цепочку классов от данного класса «наверх» (к классу, от которого он унаследован), то за *конечное число шагов* мы придем к классу **Object.** Этот путь обозначим как *«путь наверх»*. Между прочим, это означает, что все Java-классы связаны в одно дерево и корнем этого дерева является класс Object.

Будем и мы *предполагать, что наши типы связаны в дерево и что «нулевой тип» **object** является корнем этого дерева*. Как работать с деревьями, описано в следующих разделах.

Если в таком дереве построить от данного типа «путь наверх», получим *последовательность типов.* Если взять коды этих типов, получим *последовательность таблиц.* Если объединить все поля этих таблиц, получим *список «элементов данных» данного типа.* Списки элементов данных, полученные таким образом, ведут себя точно так же, как списки переменных объекта при наследовании Java-классов, — *при новом наследовании список переменных (полей) прирастает за счет нового класса (новой таблицы).*

Наследование — весьма частая логическая операция. Покажем на примере, когда это бывает и как с этим работать.

Рассмотрим тип «Сотрудник (employee)». Пусть он унаследован от типа «Человек (person)», а тот в свою очередь унаследован от типа «Объект (object)» (в скобках указан код типа, перед скобками — имя типа). Для того чтобы найти все элементы данных данного объекта (рис. Ю):

1. Найдем его номер.
2. Найдем запись в таблице **object** для этого объекта.
3. Найдем его тип.
4. Найдем путь наверх по таблице. Предположим, что для указания наследования используется поле **parent.**

5. Перечислим все таблицы по *пути наверх.*
6. Выпишем все поля этих таблиц, кроме поля **id,** начиная с таблицы **object** *вниз по пути.*

Четвертоерешение. Множественноенаследование(наследование одного нового класса от более чем одного старого класса) отсутствует в языке Java, но применяется в языке C++. Ниже мы добавим в нашу модель множественное наследование и рассмотрим случаи, когда оно полезно.

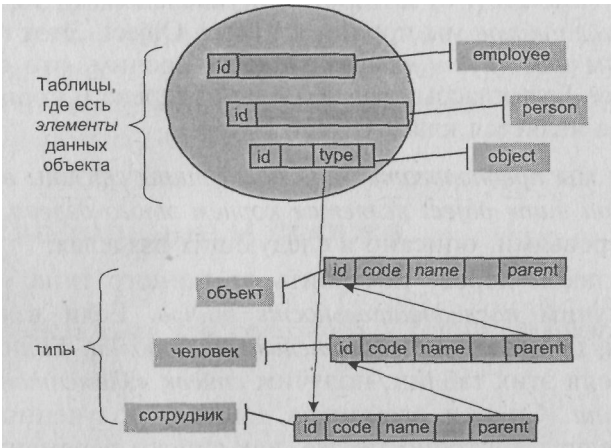

Рис. 10. Как найти все элементы данных

14. Деревья на таблице

Чтобы строки одной таблицы связать в дерево, достаточно ввести поле parent для хранения связей.

```
CREATE mytable(id INT,...,parent INT DEFAULT 0,
CONSTRAINT pk_mytable PRIMARY KEY (id),
CONSTRAINT fk_parent FOREIGN KEY(parent)
REFERENCES mytable
ON DELETE CASCADE)
```

Опция ON DELETE CASCADE гарантирует, что вместе с удалением узла из дерева (из таблицы) будут удалены все узлы, лежащие ниже его в дереве. Пример такого дерева показан на рис. 11. Корнем дерева является запись, у которой **id=0**.

Можно добавить еще одно поле (**parent2**) и построить на той же таблице еще одно дерево (рис. 12). Единственное ограничение: у всех таких деревьев один и тот же корень (id=0).

Примечание. *Выше при сборке полей мы исключили поле id как «служебное». Поля* ***parent, parent2, object::type*** *и* ***им*** *подобные тоже являются служебными, и при сборке их тоже надо исключить.*

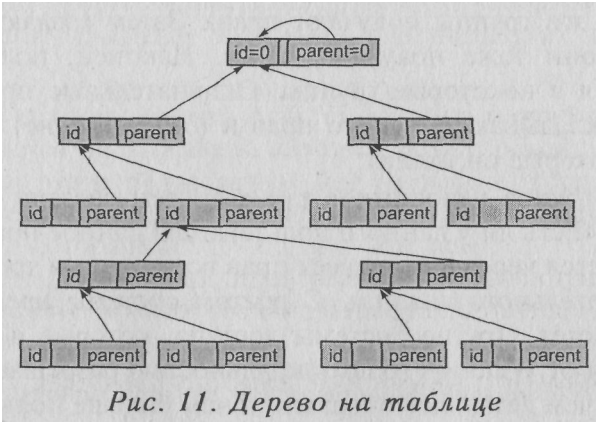

Рис. 11. Дерево на таблице

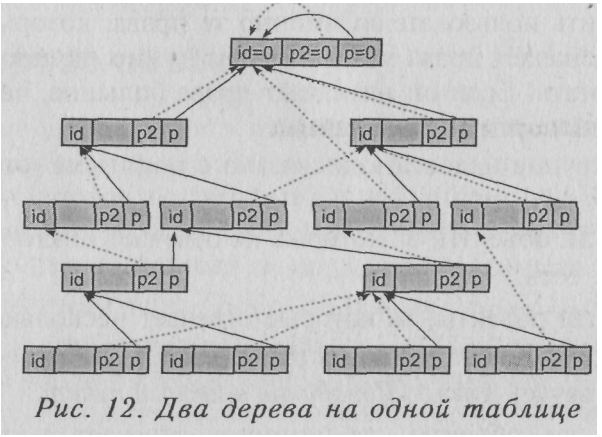

Рис. 12. Два дерева на одной таблице

Архитектор

15. Управление доступом

Все известные мне системы управления доступом построены по реляционному принципу. Наиболее известным примером этого подхода является Windows. Здесь создаются группы пользователей, эти группы получают права. Затем создаются пользователи, они тоже получают права. Наконец, пользователи включаются в некоторые группы. Окончательные права пользователя складываются из его прав и (объединение) прав всех групп, в которые он входит.

Первая группа недостатков таких систем связана с ответом на вопрос «Есть ли у данного пользователя данное право?». Вопрос решается через анализ всех прав всех групп и требует долгого и тщательного анализа. С ростом системы время такого анализа растет. Другие системы доступа, которые я встречал, отличаются от Windows только детальностью разрешаемых операций. Но чем детальнее операции и чем больше пользователей и групп, тем больше время анализа. Администратору все труднее назначить пользователю именно те права, которые нужно. Если он назначает права меньшие, чем нужно, пользователь не может работать. Если он назначает права большие, чем нужно, пользователь портит чужие данные.

Вторая группа недостатков связана с понятием «ответственность».

- Есть ли объекты, за которые не отвечает никто? Как правило, есть.
- Есть ли объекты, за которые отвечает несколько пользователей? Как правило, есть. Отвечают многие — значит, не отвечает никто. *Попадание в первый пункт.*
- Есть ли объекты, за которые отвечает пользователь (один), который в системе есть, а на работе отсутствует надолго? Как правило, есть. Фактически это *попадание в первыйпункт*

Безответственность приводит к тому, что в системе накапливаются откровенно плохие данные (мусор). Но удалить их нельзя — принять решение об этом может только тот, кто отвечает за данный объект — а такого человека в системе нет. В момент, когда мусора становится слишком много, система умирает — ею перестают пользоваться. Фильтры и прочие хитрости могут отодвинуть момент смерти, но ненамного.

Все эти недостатки существующих систем управления доступом можно сформулировать по-другому — как *требования к новой системе управления доступом,* лишенной этих недостатков. Итак:

1. Для каждого объекта в системе быстро и однозначно определяется пользователь, который за этот объект отвечает.
2. Если этого пользователя нет на работе, быстро и однозначно определяется следующий по порядку пользователь, который за этот объект отвечает.
3. Один выделенный пользователь (администратор) всегда на работе, и, если все остальные пользователи не на работе, за все отвечает он.

Наша модель данных этим требованиям удовлетворяет, об этом ниже.

16. Дерево объектов

Мы уже умеем строить *«дерево на таблице»* и уже построили *дерево наследования* на таблице type. Кроме того, мы уже обнаружили некоторое сходство между таблицами type и object. *Попробуем построить дерево* на таблице object и обеспечить на нем свойства, перечисленные в предыдущем разделе. Для этого предположим, что:

1. Пользователи тоже объекты.
2. При входе в систему пользователь получает в свое распоряжение поддерево дерева объектов, корнем которого является он сам. Это поддерево называется «зоной ответственности данного пользователя».
3. Корень дерева (выделенная вершина) соответствует выделенному пользователю — администратору.

Теперь, с какого объекта мы бы не начали, всегда существует единственный *путь наверх* от вершины к корню. Первая вершина на этом пути, которая соответствует пользователю, — пользователь, ответственный за этот объект. Если его нет на работе — берем следующую вершину-пользователя по пути наверх. Если и его нет — берем следующую... Последняя вершина — корень, соответствует администратору, а это пользователь, который всегда на работе. Что и требовалось доказать.

Если на предприятии существует иерархия управления (а обычно она существует), то отношения подчинения на производстве и отношения пользователей в дереве должны совпадать — тогда процесс распределения ответственности будет *естественным*.

В зоне ответственности можно удалить любой объект (кроме корня), можно изменить любое поле объекта (кроме кода объекта-корня). Под любым из объектов зоны можно создать новый объект.

17. Создание объектов

Когда создается новый объект, первое, что надо сообщить системе, — какого типа будет объект. Тип можно выбрать из таблицы типов. Однако с ростом системы таблица типов растет, и список типов становится слишком длинным. Кроме того, представьте себе, что у вас есть в системе пользователь-бухгалтер, пользователь-конструктор и пользователь-технолог. Конструктор ничего не знает о материалах, бухгалтер ничего не знает о деталях, технолог ничего не знает о платежах. Зачем им в *списке «документов, которые можно создать»,* документы, которые они заполнять *не умеют*? Разумно ограничить такие списки. Но как?

Давайте унаследуем от типа Пользователь типы Бухгалтер, Конструктор и Технолог. Создадим типы Папка Бухгалтера, Папка Конструктора и Папка Технолога. Предположим, что Бухгалтерские документы могут появляться только в Папке Бухгалтера, конструкторские только в Папке Конструктора, а технологические — только в Папке Технолога (рис. 13).

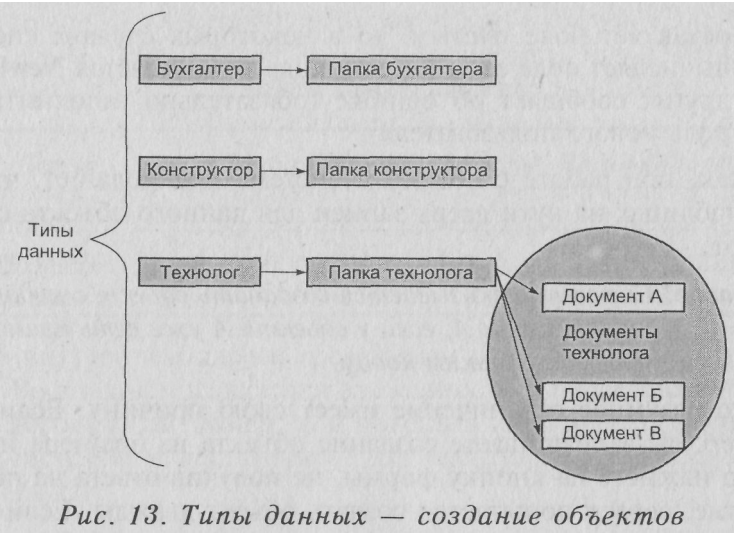

Рис. 13. Типы данных — создание объектов

Фактически, указанные здесь связи образуют дерево на таблице типов. Но это не дерево наследования, а другое (второе) дерево. Мы будем называть его *деревом вложенности*.

Примечание 1. В программе FTS *дерево вложенности* использует поле **parent,** а *дерево наследования* — поле **parent2.**

Итак, пользователь А может создать под объектом Б типа ТБ новый объект В типа ТВ тогда и только тогда, когда:

1. Объект Б входит в *зону ответственности* пользователя А.
2. Тип ТВ в дереве вложенности является *прямым потомком* типа ТБ (или ТВ это особый тип **Папка,** который можно создавать всюду).

Примечание 2. Итак, у нас есть уже три особых типа, которые есть в любом варианте модели FTS. Это Объект **(object),** Пользователь (**xuser**) и Папка **(folder).**

Что происходит, когда система создает новый объект типа Т?

1. Система генерирует новый номер объекта **NewID**.
2. Вычисляется путь наверх из Т в корень дерева наследования. Во все таблицы данных на этом пути от корня к Т вставляются записи с этим номером: **id=NewID.**
3. У пользователя есть возможность вставить значения во все (неслужебные) поля этих таблиц. Если пользователь

оставляет поле пустым, то в некоторых случаях система вычисляет поле сама (вместо кода записывается **NewID**), в других сообщает об ошибке (обязательно заполнить пароль нового пользователя).

Итак, при работе с объектом система предполагает, что во всех таблицах на пути вверх записи для данного объекта существуют.

Внимание! *Система отказывается создать объект с кодом X под объектом A, если у объекта A уже есть прямой потомок с таким кодом.*

Это разумное ограничение имеет свою причину. Если, например, вы запрашиваете создание объекта из браузера и вторично нажмете на кнопку формы, не получив ответа на первое нажатие, сервер попытается создать объект дважды. Если поле кода пусто, он создаст два объекта с разными номерами и равными им кодами. Если поле кода не пусто, то второй объект не будет создан и вы получите сообщение об ошибке вроде *«такой объект под данным родителем уже существует»*.

18. Гибкие деревья

Во всех трех деревьях (объектов, вложенности типов, наследования типов) разрешается перемещать узлы от одного родителя к другому, при этом все поддерево следует за своим корнем. Единственное условие: *не должен возникать порочный круг,* а для этого *нельзя* корень поддерева перемещать *под вершину того же поддерева*. Например на рис. 14 новым родителем вершины A может быть любой узел дерева кроме узлов A, B, C.

Внимание! *При перемещении вершины в дереве наследования у некоторых объектов изменится путь наверх и в некоторых таблицах на этом пути не будет записей для этих объектов. Система следит за этим и добавляет такие записи.*

Как сообщить системе имя нового родителя? Для этого существуют *два вида адресов:* внешние и внутренние. В следующей таблице показана первая буква адреса.

Таблица 5

Первая буква адреса

Дерево	*Внутренние адреса*	*Внешние адреса*
Объектов	A	a
Вложенности	T	t
Наследования	H	h

Во внутреннем адресе *после буквы* идет абсолютный номер узла. Во внешнем адресе для вложенности и наследования тоже идет абсолютный номер, для объектов идет номер, вычисляемый системой и достаточный для идентификации узла в данный момент.

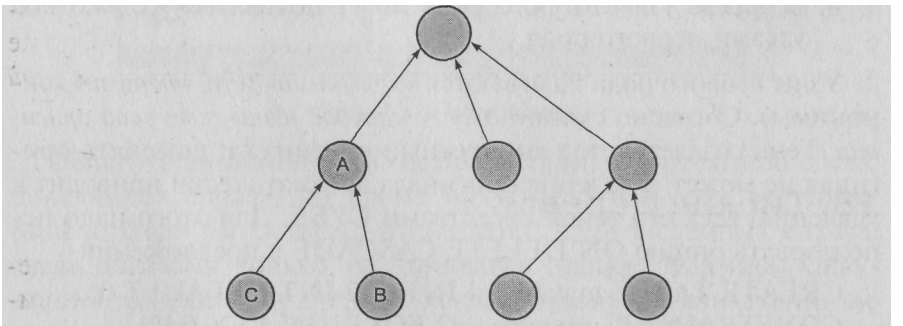

Рис. 14. Новым родителем узла A не могут быть узлы A, B, C

19. Ярлыки, тени, оригиналы

Вариант модели, который описан до сих пор, не знает промежуточных ситуаций. Если пользователь видит объект, он может его удалить, заменить, переместить. Однако есть случаи, когда один пользователь А хочет дать другому пользователю Б возможность видеть объект X, но не право изменять этот объект.

Зона видимости пользователя Б (по определению) состоит из всех объектов, которые он видит. *До сих пор зона* видимости пользователя совпадала с его зоной ответственности. *Теперь* у

нас появится возможность *расширить зону видимости за пределы зоны ответственности.*

Рассмотрим таблицу Т и дерево Д на этой таблице. Строки таблицы представляют узлы дерева. Раньше у нас не было иных узлов, теперь они появятся. Поэтому узлы, известные до сих пор, назовем узлами первого рода. Предположим, нам разрешено добавлять в дерево узлы «второго рода» по следующим правилам:

1. Каждый такой узел имеет свой абсолютный номер.
2. Каждый такой узел присутствует как строчка в таблице Т.
3. Каждый такой узел указывает на некоторый узел первого рода («оригинал»). Для этого в таблицу добавлено поле **id2.** Оно указывает на оригинал. Если в этом поле стоит О, то эта строчка — узел первого рода. Нулевой узел оригиналом не бывает.
4. В дереве узлы второго рода могут появляться только под узлами первого рода.

Узлы второго рода называются *«ярлыками»* или *«тенями»* (синонимы). Согласно сказанному, *у каждой тени есть свой оригинал.* Тень создается под конкретный оригинал и поменять оригинал не может. Удаление оригинала автоматически приводит к удалению всех его теней средствами СУБД. Для этого надо использовать опцию ON DELETE CASCADE в предложении

```
CREATE TABLE mytable(id INT, id2 INT DEFAULT 0,...
CONSTRAINT fk_mytable_id2 FOREIGN KEY (id2)
REFERENCES mytable ONDELETE CASCADE)
```

Для единообразия мы будем называть оригиналами все узлы первого рода, независимо от того, есть у них тени или нет.

20. Как показать дерево

Проводник в системе Windows (MS Explorer) является ближайшим примером того, как показать дерево пользователю. Однако этот пример обладает рядом существенных «нелогичностей», которых нам в своем интерфейсе хотелось бы избежать. Так (рис. 15) выглядит окно Проводника (на моем компьютере).

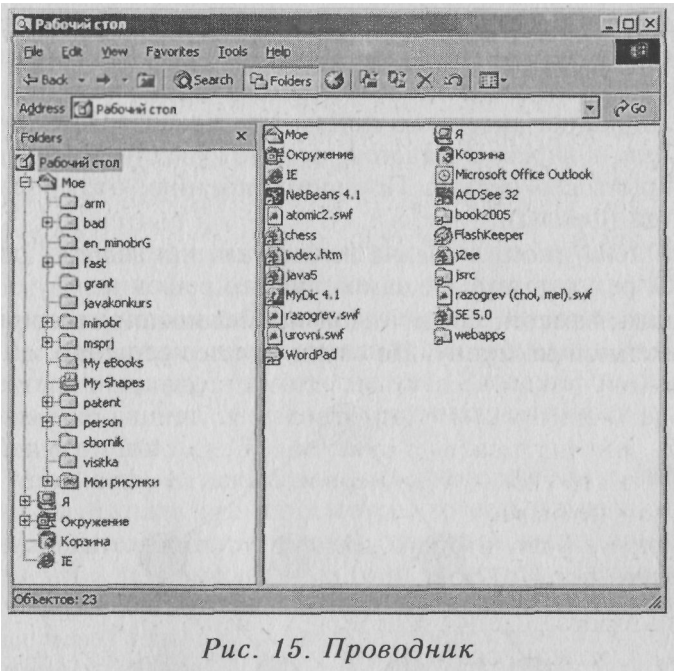

Рис. 15. Проводник

Обратите внимание, что «дети» вершины, отмеченной слева, появляются справа, но кроме них справа есть еще вершины. Можно было бы сказать, что из детей, перечисленных справа, слева показаны только «контейнеры» (папки, Корзина, Окружение), а файлы и ярлыки контейнерами не считаются, но можно ли назвать папкой Internet Explorer (IE)? Здесь очевидны следующие логические несообразности.

1. Если некоторые элементы отображены справа — зачем их еще раз отображать слева?
2. Если другие элементы отображены справа, а слева нет, то почему?
3. Двойной клик на контейнере справа открывает этот контейнер. Двойной клик на ярлыке контейнера тоже открывает этот контейнер — в чем разница?
4. Каждый элемент интерфейса имеет свои свойства, но чтобы их увидеть, надо кликнуть правой кнопкой, а затем выбрать правильную строчку из всплывающего меню — зачем так длинно?

5. Начальное состояние дерева при открытии Проводника состоит из произвольно выбранного разработчиками Windows набора узлов.
6. Если папка содержит файлы, но не содержит ни одной папки, в дереве считается, что этот узел не имеет детей, и открыть его нельзя. По-моему мнению, этот узел имеет детей (файлы).
7. Клик на икону и на название узла приводит к одинаковым результатам. Не самое лучшее решение.
8. Ярлык заметен среди узлов по маленькой стрелочке, наложенной на икону. Не самое лучшее решение.

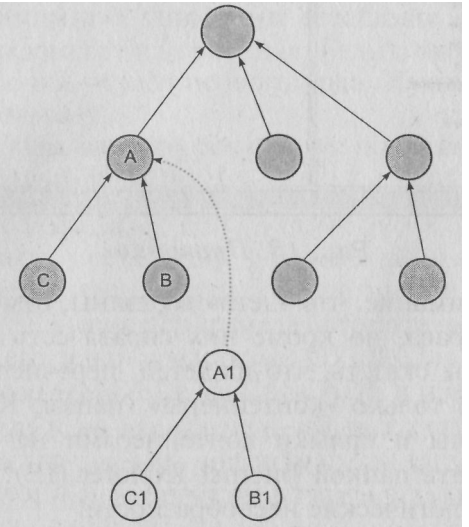

Рис. 16. Тени и псевдоузлы

На наш взгляд, интерфейс для отображения дерева должен быть построен следующим образом:

1. Слева в начале работы должна отображаться только одна вершина (какая — зависит от того, какой пользователь работает).
2. Узлы любого типа (и оригиналы, и ярлыки, но не элементы данных), которые содержатся в данном узле A, считаются его «детьми» в дереве.

3. Кликнув на элемент в дереве (на его названии) и, соответственно, выделив его, справа получаем его состояние, в котором видны все элементы данных узла и все операции, разрешенные нам над этим узлом.
4. Если у оригинала A есть дети (B, C ...), то у его ярлыка (A1) тоже будут дети, мы называем их «псевдоузлы» (B1, C1 ...) (пример на рис. 16). Кликнув на тень или псевдоузел, справа получим элементы данных оригинала (соответственно элементы данных потомков оригинала).
5. Тень отличается от псевдоузла тем, что тень можно удалить (при этом пропадают все псевдоузлы). Псевдоузел, взятый отдельно, удалить нельзя. Удаление тени на оригинал не влияет. Удаление оригинала автоматически удаляет все тени.
6. У тени есть «теневые данные», которые «затеняют» данные оригинала при просмотре, то есть замещают их для пользователя. Пользователь, который видит тень, может изменить теневые данные (они хранятся отдельно в каждом ярлыке). Другие элементы данных оригиналов, видимые через тень или псевдоузел, через тень или псевдоузел изменить нельзя.
7. Ярлык виден по измененному шрифту названия (стиль: *италик).* Икона при этом не меняется.
8. Клик на икону узла (в отличие от клика на названии узла) дает справа не элементы данных и операции узла, а функции узла. ***Операции*** *одинаковы* для всех узлов данного дерева. Но каждый *тип* узлов имеет свой *список функций,* как и свой *список элементов данных.* Список функций *растет по правилам наследования,* подобно списку элементов данных. У каждого типа своя икона. Хотя одну икону разрешается использовать для нескольких типов.

Примеры деревьев в интерфейсах FTS подробно даны на диске, поэтому здесь их нет.

Примечание. *Пользователь видит дерево объектов начиная с себя, а дерево вложенности — со своего* ***типа.*** *Дерево наследования он видит целиком.*

Знаток

21. Задача разузлования

Предположим, конструктор K1 разработал узел X, а конструктор K2 решил включить его один раз в изделие C1 и два раза в изделие C2. Как отобразится это в FTS? Решение показано на рис. 17.

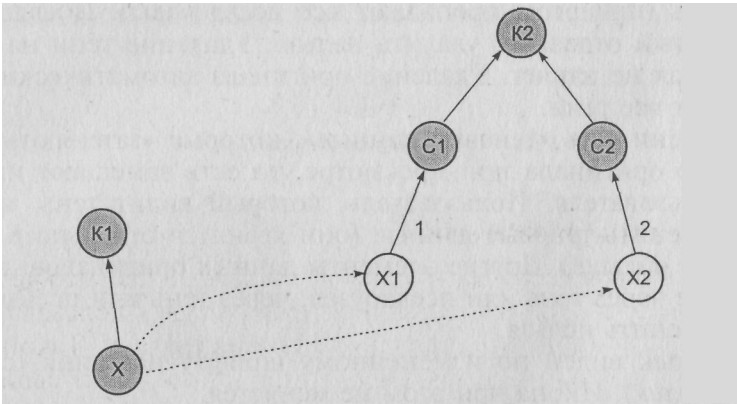

Рис. 17. Конструктора и их изделия (цифры 1 и 2 показывают «входимость»)

Один оригинал X имеет две тени X1 и X2 соответственно. X1 и X2 находятся в зоне ответственности пользователя K2, а X в зоне его видимости. Если K1 изменит X, то K2 увидит эти изменения в X1 и X2, как только обновит свое дерево (еще раз прочитает его узлы с сервера). Если под X есть еще узлы (рис. 18), то под X1 и X2 появятся соответствующие псевдоузлы. K2 не может их удалить, поэтому они вне его зоны ответственности, так же как и их оригиналы. Но они в зоне его видимости (так же как и их оригиналы).

Теневые данные ярлыка необходимы для того, чтобы показать «входимость» узла в изделие. Если в одной точке дерева узел используется *один раз,* а в другой *два раза,* то эти числа (1 и 2) надо хранить именно в ярлыке.

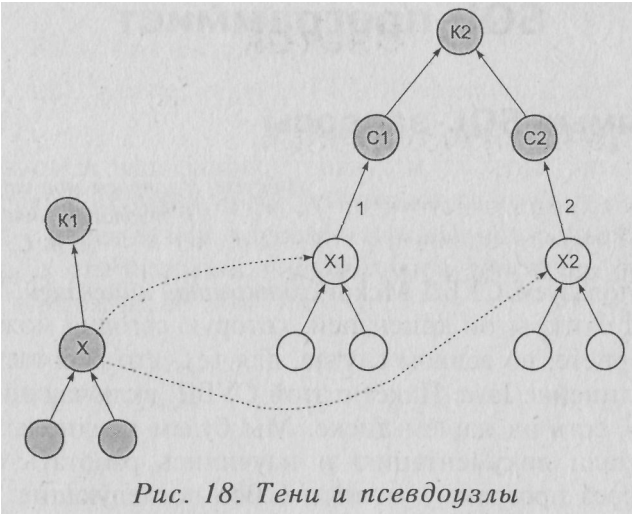

Рис. 18. Тени и псевдоузлы

Слепые узлы и ярлыки

Иногда появляется необходимость сделать так, чтобы узел, находящийся в зоне ответственности некоторых пользователей показывал своих потомков только администратору. Такой узел я называю «слепым», подразумевая, что «он не видит своих детей». Чтобы сообщить системе, что узел X слепой используется поле ABSTR (boolean) в таблице OBJECT. Это, поле по умолчанию FALSE, по умолчанию узел НЕ слепой.

Если узел слепой, то и все его ярлыки не показывают его детей тоже.

Есть возможность объявить слепым один конкретный ярлык узла. Тогда этот ярлык не будет показывать детей узла, независимо от того, является ли слепым сам узел.

Слепоту узла задает/изменяет только администратор.

При создании ярлыка слепота узла дублируется слепотой ярлыка.

Слепота при создании ярлыка задает специальный CHECKBOX при кнопке SHORTCUT.

Слепоту узла можно задать той же кнопкой, если поле адреса оставить пустым.

SQL-программист

22. Прямые SQL-запросы

И вкусы, и запросы мои странные —
я экзотичен, мягко говоря.
В. С, Высоцкий

Мы используем СУБД Mckoi *с открытой лицензией.* Это *лучшая* СУБД с открытой лицензией, которую сегодня можно найти в Интернете, во всяком случае, для тех, кто работает в продуктовой линейке Java. Пакет с этой СУБД, включающий документацию, есть на нашем диске. Мы будем предполагать, что вы прочитали документацию и научились работать с Базой данных через программу «клиент». Все последующие SQL-запросы мы рекомендуем вам проверить подобным образом.

Внимание! Если http-сервер (Apache Tomcat) включен, то он монополизирует БД и клиенту с ней связаться не удастся. Поэтому будем считать, что сервер остановлен и что клиент с БД связался.

Прежде всего отмечу, что информация в БД разбита на схемы. Чтобы увидеть список схем, используйте команду SHOW SCHEMA. Схемы SYS_INFO и SYS_JDBC хранят служебную информацию. Схема APP используется по умолчанию. В текущей реализации FTS на одном сервере возможно несколько подсистем, каждая подсистема имеет отдельное имя, это имя совпадает с именем соответствующей схемы в БД. Например, если вы обнаружили в нашей БД схему с именем ABC, это означает, что с этой базой и на этой схеме работает FTS-подсистема с именем ABC. Чтобы работать с информацией этой схемы, используйте команду SET SCHEMA <имя схемы>.

Чтобы увидеть все таблицы базы, используйте команду SHOW TABLES.

Чтобы увидеть все поля таблицы, используйте команду DESCRIBE <имя таблицы>.

Чтобы увидеть все данные таблицы, используйте команду SELECT * FROM <имя таблицыX

Как узнать номера, коды, имена всех объектов типа **folder** (Папка)? Самый надежный путь:

```
SELECT o.id, o.code, o.name FROM object o, folder t WHERE t.id=o.id
```

При этом в ваш запрос попадут и те узлы, типы которых унаследованы от **folder,** и те, чей тип когда-то был унаследован **folder,** но с переменами в дереве наследования больше таким не является, а специальной чисткой типа никто не занимался. (Чистка возможна, об этом ниже.)

Знаток

23. Древовидные типы

Вычислительные объекты (например, объекты языка Java), как правило, имеют «жесткие типы». Я имею в виду тот факт, что тип, заданный в момент создания объекта, в течение жизни объекта изменить невозможно. Использование объекта через интерфейс или по типу *на пути наверх* ничего не меняет.

Однако концептуальные типы устроены иначе. Возьмем, например, объект **Вася Иванов.** Вася Иванов учится в университете на последнем курсе. Поэтому мы можем считать, что его тип **Студент,** что этот тип унаследован от типа **Человек,** а тот в свою очередь — от типа **Объект.** Таким образом, как показано на рис. 19, таблицы объекта описаны областью 1.

Мы уже отмечали, что в FTS дерево наследования типов гибкое (можно поменять родителя), чего нет ни в Java, ни в других языках. Но сейчас речь не об этом.

Представьте себе, что **Вася Иванов,** продолжая учебу в университете, одновременно поступил на работу на завод. Теперь

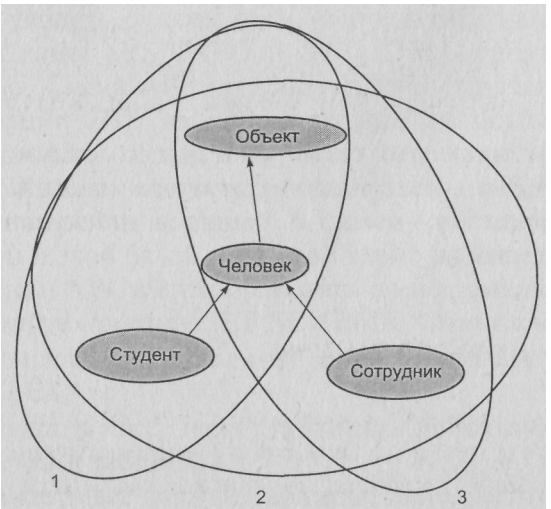

Рис. 19. Древовидный тип

мы можем присвоить этому объекту тип **Сотрудник.** Однако как быть с тем, что накоплено о нем как о студенте?

Понятно, что в дереве наследования ни **Сотрудник** не унаследован от Студента, ни студент от сотрудника, но оба они от типа **Человек.** Старые данные остаются в системе. Теперь таблицы объекта описываются областью 2. Наконец, **Вася Иванов** закончил университет. Теперь он не Студент, но только **Сотрудник.** Теперь данные о нем как о студенте надо уничтожить. Теперь его таблицы описываются областью 3.

Чтобы различить эти три случая, мы введем следующие определения.

Тип-множество определяет, *что хранит объект* (какие элементы данных).

Тип-центр определяет, *что демонстрирует объект по умолчанию* (элементы данных, функции). Тип-центр относится к тип-множеству как элемент к множеству.

Все *тип-центры* соединены в дерево (это наше *дерево наследования).*

Тип-множеством может быть *любое поддерево этого дерева, содержащее его корень.*

Каков же тип-центр нашего объекта в каждом из трех случаев? В первом случае это, скорее всего, **Студент,** в третьем, скорее всего, — **Сотрудник.** Во втором случае это может быть или **Студент,** или **Сотрудник,** или **Человек,** но не **Объект.**

Внимание! *По некоторым соображением (их я изложу ниже) в системе есть только один объект типа Объект — это нулевой объект (корень дерева объектов, Администратор).*

24. Зона компетенции

Если объект А попал в зону ответственности пользователя Р, то, по общему смыслу, Р *может* изменить его *тип-множество* и/или *тип-центр. Сужение* тип-множества и/или *перемещение* тип-центра (или короче: ***сужение и перемещение типа)*** *абсолютно безопасно,* и мы можем такие операции не ограничивать. Од-

нако на право расширять тип-множество надо наложить одно разумное ограничение.

Представьте себе, что конструктор решил расширить тип-множество так, чтобы в объект А входила бухгалтерская информация, в которой он ничего не понимает. *Такое расширение типа надо запретить.* Но как *отличить* разрешенные *расширения* от запрещенных? На этот счет, после долгих экспериментов, нам удалось сформулировать два правила.

1. *Администратор* может осуществлять *любое* расширение типа.
2. Другой пользователь может расширить тип объекта в том и только в том случае, если при этом *объединение всех тип-множеств объектов из его зоны ответственности* не увеличится. Такое объединение мы будем называть *зоной компетенции.* И правило будет звучать так: *зона компетенции пользователя при расширении типа не должна увеличиваться.*

В чем смысл зоны компетенции? Если объединить списки элементов данных всех объектов зоны ответственности данного пользователя, то можно «очертить его компетенцию». Обычно конструктор не понимает в бухгалтерской отчетности, но данный конкретный конструктор может понимать. Если у него в зоне ответственности есть хотя бы один бухгалтерский объект — значит да, понимает. Если нет — значит нет, не понимает. Кто это может определить и поправить? Если начальник сам понимает в бухгалтерских документах, он может переместить такой документ в зону подчиненного — и *зона компетенции подчиненного соответственно увеличится* (рис. 20).

Примечание. Зона ответственности может быть большой, а зона компетенции предполагает перебор всех таблиц от корня дерева наследования по каждому объекту, что довольно дорого. Поэтому *в действующей версии* FTS принят компромиссный вариант: *учитывается не вся область ответственности, а только та, что видна в дереве в данный момент* (часть узлов в дереве может быть свернута, а их потомки, соответственно, скрыты).

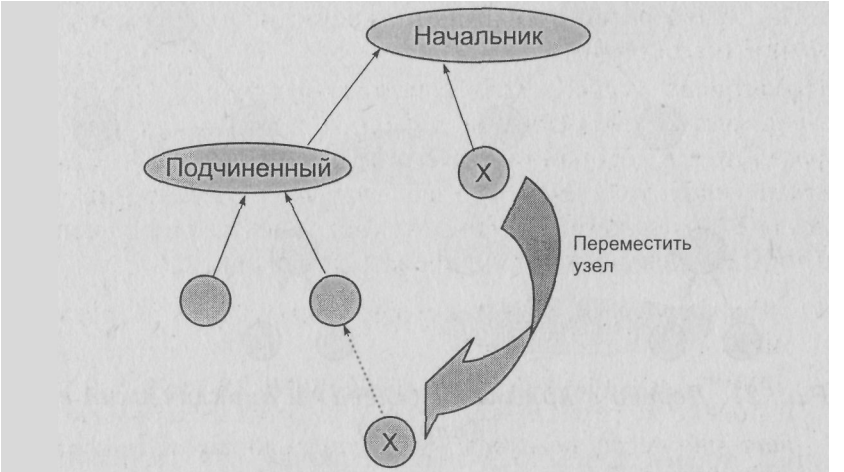

Рис. 20. Чтобы увеличить область компетенции, можно переместить узел

25. Множественная вложенность

Возможность добавить в дерево ярлык узла превращает дерево (ориентированное от корня) в (ориентированный) ациклический граф (с одним источником). Но есть отличие: в ациклическом графе все ребра, входящие в узел, равноправны. (Замена ориентации всех ребер на противоположную превращает источник в сток, но это только вопрос терминологии.)

В нашем «дереве с ярлыками» одно из ребер выделенное: его удаление удаляет узел и все остальные ребра. Удаление невыделенных ребер побочных эффектов такого рода не имеет.

Добавление ярлыков в дерево вложенности дает интересный эффект. Предположим, у нас есть тип **Учитель физики** и тип **Учитель химии** (рис. 22). Учитель физики создает тесты по физике, учитель химии — тесты по химии. Предположим, у нас появился тест, который хотели бы создавать и тот и другой. Без ярлыков это было бы невозможно, у каждого типа был бы только один родитель. Добавление ярлыков решает эту проблему (смотри рис. 22).

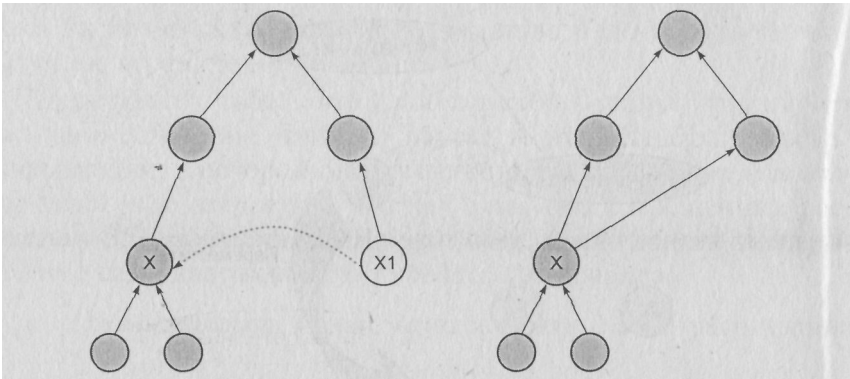

Рис. 21. Дерево с ярлыками (слева) и ациклический граф (справа)

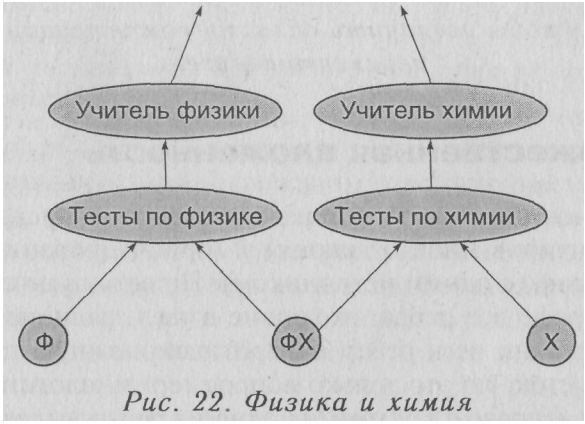

Рис. 22. Физика и химия

26. Множественное наследование

В языке Java множественного наследования нет, в языке C++ — есть. Покажем пример, когда множественная вложенность хорошо работает.

Предположим, завод выполняет заказы на несколько видов работ, причем каждый вид — особый тип, со своим списком параметров (рис. 23).

В заказ детальный есть возможность включить параметры работы А и/или работы Б.

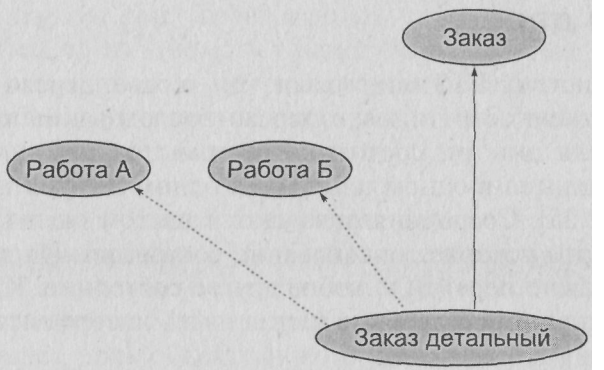

Рис. 23. Множественное наследование

Вспомним, что вычислительные объекты имеют данные *(member variables)* и функции *(member functions).* Список допустимых данных и функций определяется типом. При наследовании списки данных удлиняются, списки функций тоже. При этом списки сохраняют внутренний порядок элементов. Правило объединения списков при множественном наследовании показано на рис. 24. Окончательный список строится так: строится *главный путь наверх* (этот путь не содержит ярлыков, он *единственный).* Затем к нему присоединяются элементы всех *остальных путей наверх,* если они еще не включены.

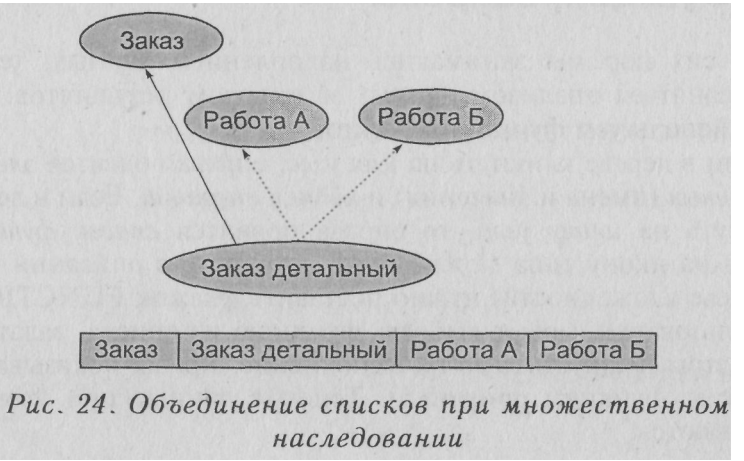

Рис. 24. Объединение списков при множественном наследовании

27. Три дерева

Итак, систему FTS описывают три дерева: дерево объектов, дерево вложенности типов и дерево наследования типов. Для пользователя это три состояния системы, и для него система всегда находится в одном и только в одном из этих трех состояний (рис. 25). Состояния отличаются цветом (желтый, синий, зеленый), это ускоряет опознавание состояния. Из любого состояния можно перейти в любое другое состояние. Кнопки для этого есть над и под деревом (левая часть интерфейса).

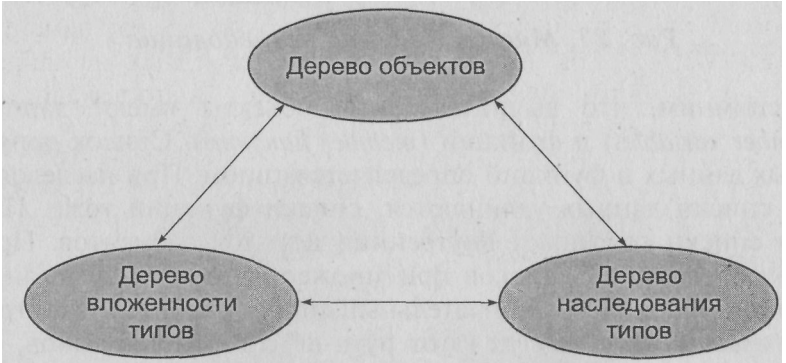

Рис. 25. Три состояния системы

28. Функции, запросы

До сих пор мы занимались накоплением данных, теперь пора заняться анализом данных и выводом результатов. Для этого используем функции объектов.

Если в дереве кликнуть на *имя узла,* справа появятся *элементы данных* (имена и значения) и *кнопки операций.* Если в дереве кликнуть на *икону узла,* то справа появится *список функций.* Чтобы на икону типа можно было кликнуть, в описании типа (в дереве вложенности) нужно поставить флажок FUNCTION

Пользователь может выбрать функцию из списка, задать ей параметры и запустить ее на выполнение. Ярлык показывает и запускает функции оригинала. Теневые данные при этом не учитываются.

Кроме параметров, передаваемых непосредственно перед выполнением, функция может использовать данные, хранящиеся в объекте, данные о местоположении объекта в дереве объектов и любые другие данные, доступные в системе.

Выводные данные обычно имеют вид одной или нескольких таблиц. Конкретный вид таблиц зависит от функции.

Итак, список функций узла определяется типом узла. В начальную конфигурацию FTS включен особый тип query. Если данные можно получить средствами SQL, то соответствующий запрос составляется SQL-программистом и записывается в соответствующее поле объекта нового объекта типа query. Чтобы пользователь мог запустить такой запрос, но не мог изменить его, ему в зону ответственности передается ярлык на запрос. Оригинал находится у администратора или у пользователя, выполняющего работу SQL-программиста.

SQL-программист

29. Объект «Запрос»

В этом объекте есть поле **query::body**, в которое записывается SQL-запрос *(тело запроса* или *тело)* и поле **query::pardesc** (parameters' description), в которое записывается *подсказка о параметрах.*

Параметры позволяют *модифицировать тело* непосредственно перед выполнением. Перед анализом ситуации пробелы в начале и в конце строки параметров отсекаются. Далее возможны три варианта.

1. Строка параметров P пуста. Никаких модификаций тела.
2. Строка P не пуста и не содержит знак «равно» (=). В этом случае система находит в теле все вхождения символа @ и заменяет его на P.
3. Строка содержит по крайней мере один знак =. В этом случае система делит P на части вида A=a; B=в; ... Анализ идет слева направо, ищется равенство, затем точка с запятой, затем равенство и т. п. Далее в теле ищутся все вхождения строки вида @A#, где A — первое из имен в P. Все эти вхождения заменяются на первое найденное значение a и т. п.

Теперь, получив запрос, система выполняет его и, в случае успеха, получает массив записей. Теперь она должна преобразовать этот массив записей в таблицу (одну или несколько) и выдать это.

В простейшем случае столбцы таблицы те же, что в запросе между SELECT и FROM. SELECT id, code, name FROM object выдаст таблицу вида:

id	code	name

Строки таблицы будут соответствовать полученным записям. Можно прямо в запросе задать иные (например, русские) имена столбцов, например запрос

SELECT id "Номер", code "Код", name "Наименование" FROM object
выдаст таблицу вида:

Номер	**Код**	**Наименование**

В более сложном случае можно «разрезать» одну таблицу на несколько так, чтобы в каждой таблице значения первых столбцов были постоянными. Например, таблицу

Отдел	Сотрудник	**Зарплата**
Технологи	Иванов	5000
Технологи	Петров	6000
Конструктора	Сидоров	7000
Конструктора	Лебешев	8000

имеет смысл разрезать на две — по постоянству первого поля, а само это поле вынести в заголовок таблицы, вот так:

Отдел = Технологи

Сотрудник	**Зарплата**
Иванов	5000
Петров	6000

Отдел = Конструктора

Сотрудник	**Зарплата**
Сидоров	7000
Лебешев	8000

Для того чтобы результат соответствовал интуитивно правильному, надо отсортировать записи командой ORDER BY <имена столбцов, выносимых в заголовок>.

Для определения, сколько именно столбцов надо проверять на постоянство, используется поле **query::headlevel**. По умолчанию в этом поле хранится 0.

В системе есть возможность сгенерировать тело запроса автоматически, а потом отредактировать вручную и записать результат в объект. Для запросов эта возможность открывается справа при нажатии на икону объекта. Начать надо с того, чтобы сообщить генератору имя таблицы, с которой начинается выполнение запроса. Далее появится возможность увидеть поля таблицы и выбрать нужные, увидеть связанные таблицы и их поля и т. п. Есть возможность задать альтернативное имя столбца и его участие в ключах сортировки.

Внимание! *При использовании генератора запросов исходной таблицей по умолчанию считается таблица* ***текущего корня дерева наследования.*** *О перемене корня дерева — в следующем разделе.*

Знаток

30. Работа с деревом

Для того чтобы ускорить работу с деревом, есть несколько интересных приемов. Кнопки справа (Н и В) позволяют сделать текущий узел корнем (Н = новый корень) и, наоборот, вернуть корень (В = вернуть корень).

В дереве наследования есть возможность создавать «абстрактные типы», которым не соответствует никакой объект и никакая таблица в базе. В дереве вложенности эти узлы тоже не появляются. Их единственное назначение — группировать узлы дерева наследования так, чтобы при развертывании узла список детей не был слишком длинным.

31. Имена полей

Если мы находимся в состоянии «Объекты», то с правой стороны мы видим элементы данных последнего активного объекта и кнопки для операций над ним.

Элементы данных образуют таблицу вида:

Имя	*Значения*
Имя1	Значение1

В простейшем случае имя имеет вид <имя таблицы>::<имя столбца в таблице>.

Этот вид удобен разработчикам, но не удобен пользователю. Поэтому в системе есть «Словарь», который хранит таблицу соответствий вида:

Внутреннееимя	*Внешнееимя*
<имя таблицы>::<имя столбца в таблице>	Полное имя для пользователя

Имена некоторых полей уже имеют в словаре перевод. Например, в словаре есть соответствие

object::code = код

Если создается новый тип или в тип добавляются новые поля, то, как правило, таких полей нет. Администратор может добавить их. Это называется «изменение имени поля» (см. ниже).

Неочевидным применением такого подхода является создание анкет (тестов). *Вопрос, на который надо ответить,* есть *«пользовательское имя поля»,* в то время как *внутренне имя поля* обычно имеет вид **q1** и т. п.

Для вопросов с выбором есть специальные приемы и механизмы (подробно описаны в соответствующей главе, далее по тексту).

32. Создание объектов

Есть несколько способов создания объектов под данным объектом (приведем номера и английские обозначения вариантов)

1. One object
2. ObjList after a(+a)
3. ObjList after children of
4. ObjTree after ObjTree from
5. ObjTree after ObjTree from (extra type)
6. ObjTreet after TypeTree (codes) from
7. ObjTreet after TypeTree (numbers) from
8. ObjList after lines from file
9. Shortcuts after q-addr (parent name=params)

По умолчанию создается один объект (1) выбранного типа. '

Если надо создать *бланки тестов для всех учеников группы,* то используется вариант (3). Указывается тип бланка и адрес папки, в которой собраны все члены группы или их ярлыки. Имена учеников **(name)** копируются в имена **(name)** бланков, а кодом бланка ставится его абсолютный номер.

Если надо сгенерировать сразу несколько бланков, но не по списку, а *выборочно,* используется вариант (2). Адреса выбранных учеников указываются через знак *плюс (+).*

Если надо сгенерировать новое поддерево объектов, «похожее» на известное поддерево объектов *(дерево задач по образцу),* используется вариант (4). Вариант (5) позволяет *расширить тип-множество* всех генерируемых объектов так, чтобы включить в него *указанный тип Т.* Пример: построение дерева задач на основе дерева сборки. Каждая деталь дает задачу, тип узла строится из *типа детали* и *расширяется* до *типа задачи.* При этом узлы в дереве-образце могут быть разных типов (деталь, сборка, материал и т. п.). При этом *тип-центр перемещается в Т.*

Если надо сгенерировать пакет бланков разного типа на основе поддерева вложенности, используются варианты (6) или (7). (6) формирует бланки по порядку кодов типов-образцов (так они показаны в дереве вложенности), (7) по порядку абсолютных номеров этих типов. Если надо импортировать узлы из текстового файла, используется вариант (8). Строки файла содержат имена и значения полей вида

<имя таблицы>::<имя поля>=<значение>;

Вариант (9) используется, когда надо в одном месте собрать ярлыки всех узлов, подпадающие под определенный запрос. Надо указать адрес узла — запроса.

Считается, что запрос имеет вид

SELECT o.id FROM object o,... WHERE ...

Строка параметров для модификации тела запроса (см. выше) записывается в имя **(name)** узла, под которым собираются ярлыки.

Пример использования: *соберите все бланки тестов, которые сдавал студент X в этом году.* В *имени* одной папки (под которой хотим собрать ярлыки) запишем X=Иванов, в *имени* другой X=Петров. Запрос может иметь вид

SELECT id FROM object WHERE name='@X#'

Внимание! Данный запрос может «захватить» больше, чем сказано выше, например, самого студента и пр.

Архитектор

33. Java = C++ — C

Есть два взаимоисключающих подхода к построению больших программных систем.

1. Все когда-либо раньше написанное должно работать в новой системе.
2. Все новое должно уничтожать старое и заменять его новым.

Первый подход характерен для Sun и Java. Второй — для C++ (C#, .NET) и Microsoft.

Наглядным подтверждением этой мысли является сам факт допущения в код C++ модулей на C. В том же духе идеология сборки программ на всех языках, на которой построена технология .NET.

Однако с ростом системы вероятность ошибки в ней возрастает. Соответственно возрастает вероятность катастрофической ошибки, и система уже не может считаться надежной. Единственным средством в таких случаях является «очистка всего», т. е. перезагрузка (см. Windows) Однако систему, которая требует перезагрузки из-за внутренних ошибок, надежной считать нельзя. Вот почему я работаю на языке Java.

34. Архитектура и процесс разработки

Во всяком программистском сообществе, даже таком продвинутом, как Java, существует набор «правил поведения» и «принципов построения», которые считаются обязательными во всяком проекте. Вот некоторые из них.

1. Код надо писать так, чтобы его мог прочитать и понять любой программист. Для этого в коде должен быть подробный комментарий. Надо использовать JavaDoc для оформления этого комментария отдельным пакетом (сайтом).
2. В ходе разработки надо использовать специализированный пакет управления версиями.

3. Сборку модулей надо осуществлять программой MAKE (вариант: ANT) и проводить ночью.
4. Управление вычислениями, управление данными и взаимодействие с пользователем надо строить раздельно. Это называется MVC = dataModel+View+Controller. Такое построение позволяет заменить, когда это потребуется, интерфейс, или СУБД, или механизм управления, не заботясь об остальных частях системы.
5. Уточнение предыдущего пункта: работу с СУБД надо строить универсально, так, чтобы замена одной СУБД на другую могла произойти без переделки кода, а только заменой одного (!) модуля.
6. Все требования и пожелания заказчика к системе должны оформляться документально. Выход версии должен сопровождаться документом о том, какие из этих требований реализованы и в каком объеме.

Все эти замечательные правила имеют смысл при одном принципиальном допущении: что разработкой системы занимается бригада программистов и что они практически равноправны. Однако в самом этом предположении есть слабое место.

Дело в том, что создание программного комплекса лишь в малой своей части есть написание кода и написание инструкций по его использованию. В основной своей части это есть *«создание системы понятий, адекватно описывающих ситуацию управленческую и ситуацию вычислительную в их взаимопроникновении и взаимосвязи»*. Написание кода и инструкций предполагает *постоянство* системы понятий. Совершенствование системы предполагает не только и не столько изменение кода и инструкций, сколько *изменение* системы понятий. Рассмотрим рис. 26.

Если кто-то из участников внес изменения в систему понятий, то эти изменения должны быть немедленно согласованы со всеми участниками. Все, что было сделано в системе понятий А после того, как появилась система понятий Б, будет выброшено как неправильное.

Один человек может работать быстро, другой медленно, но событие «элементарное согласование» во всех коллективах про-

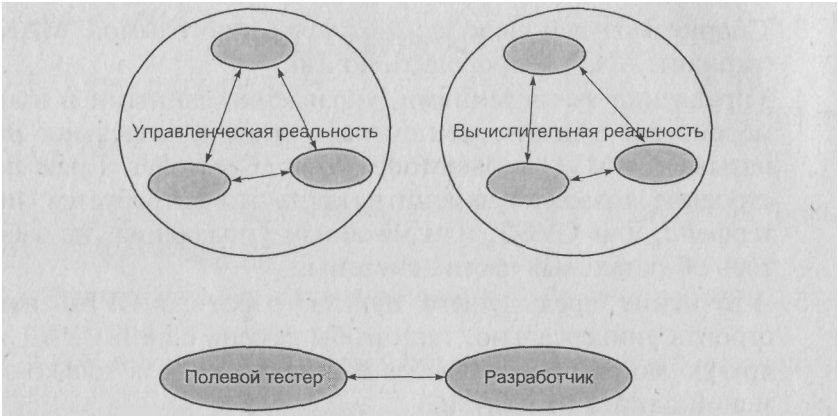

Рис. 26. Разработка коллективная и разработка парная

исходит одинаково. А вот событие «полное согласование» определяется структурой коллектива.

Если принять длительность элементарного согласования за единицу и предположить, что внутри полного согласования элементарные согласования идут последовательно, то получится, что длительность полного согласования равна сумме длительностей составляющих его элементарных согласований. В примерах, разобранных на рис. 31, нетрудно подсчитать, что длительность полного согласования при парной работе (на рис. 31 внизу) в семь раз меньше, чем длительность полного согласования при коллективной работе (на рис. 31 вверху).

Мой опыт показывает, что при парной разработке система достигает совершенства за полтора-два года. Соответственно, при коллективной работе (структура коллектива на рис. 31 вверху это еще минимальный вариант) совершенство системы будет достигнуто за десять-пятнадцать лет. Если учесть, что за пять лет в компьютерных технологиях происходит полная смена стандартов, то результат очевиден: при коллективной работе совершенство не будет достигнуто никогда.

Таким образом, единственный способ достичь совершенства в разработке систем — отказаться от коллективной разработки в пользу разработки парной. При этом один человек работает в управленческой среде и один в среде вычислительной. А если в вычислительной среде работает только один человек, то пре-

красные *шесть правил коллективной разработки* (см. выше) и все остальные правила этого рода *можно забыть*.

Какие же правила возникают в этой новой ситуации? Попробую сформулировать.

1. Используй такие имена (переменных, функций, классов и пр.), чтобы, возвращаясь на работу после двух выходных, не ломать голову над тем, *что именно имелось в виду*.
2. Копируй свой код по крайней мере на ***три*** *носителя,* находящиеся физически в разных местах — на случай пожара, ограбления и пр.
3. Имей по крайней мере ***два*** *компьютера,* на которых есть действующая версия системы, чтобы работу новой версии можно было сравнить с работой старой версии.
4. Сохраняй свою работу на жестком диске каждые пять минут, копируй на другие носители в конце рабочего дня.
5. Имей на каждом компьютере и разархивированную версию, и архивированную.
6. Настрой свою систему так, чтобы внесенные изменения можно было проверить на контрольных данных как можно быстрее (число ударов по клавишам и число мышиных кликов должно быть минимальным).
7. Проектируя систему, выбирай такие решения, чтобы они пригодились в максимальном числе случаев. Для этого умей прогнозировать запросы заказчика и решай одним решением как можно больше его задач.

Примечание для тех, кто прочитал мою книгу «Теория уровней и модель человека».

Оптимальный профиль участников парной разработки показан в таблице:

Роль	Уровни	Разогрев	Кодировка
Разработчик	67,3	Холерик	Текст, схема
Полевой тестер	58,4	Меланхолик	Картинка

При одних отклонениях от оптимума разработка замедляется, при других — становится невозможной.

Администратор

35. Расширение и развитие

Расширение системы это изменение системы объектов. Оно включает в себя:

- добавление объектов;
- уточнение элементов данных;
- уточнение типов объектов;
- изменение дерева объектов (удаление, перемещение, добавление ярлыков).

Все это — коллективное дело пользователей. Оно осуществляется в соответствии с их зонами ответственности и компетенции.

Развитие системы — это изменение системы типов. Оно включает в себя:

- добавление типов;
- изменение имени типа (код типа изменить нельзя);
- добавление поля;
- изменение внешнего (пользовательского) имени поля;
- изменение дерева вложенности и дерева типов (удаление, перемещение, добавление ярлыка).

Развитие системы — дело администратора. Никакой пользователь без администратора не имеет права на подобные операции. Администратор единолично несет ответственность за любые шаги развития.

Представим себе на минуту, что мы разрешили пользователям самим добавлять типы. Легко представить, что случится: они начнут добавлять одни и те же типы под разными именами. Внешне это будет напоминать рост раковой опухоли. Спасти такую систему уже не удастся.

Всякое требование клиента (заказчика, пользователя) по развитию системы попадает к администратору. Он должен принять решение, как осуществить адекватный шаг развития. В некоторых случаях он обращается за помощью к SQL-про-

граммисту или Java-программисту. В самых сомнительных случаях решение принимают они втроем. Однако прежде чем заказать новую функцию Java-программисту или вести новый тип, администратор должен проверить: *нет ли таких функций и таких имен в «заготовках»,* которые поставляются вместе с системой и которые Администратор может установить сам.

Мастер

36. Служебные таблицы и служебные поля

Каждому типу данных соответствует таблица. Такие таблицы называются *таблицами данных*. Кроме этого в подсистеме (в схеме) существует еще несколько таблиц. С одной из таких таблиц мы уже знакомы. Я имею в виду таблицу type, содержащую типы. Таблица translator содержит соответствие внутренних и внешних имен полей. Таблица help содержит соответствие между узлом и текстом справки по этому узлу. Таблица errlog используется для отладки Java-кода. Такие таблицы подсистемы, в отличие от таблиц данных, называются служебными. Таблица object считается одновременно служебной таблицей и таблицей данных. Структуру служебных таблиц менять нельзя.

Некоторые поля таблиц данных не показываются пользователю в качестве элементов данных, хотя система поддерживает их состояние, использует для решения внутренних проблем и значение этих полей можно увидеть, *если знать как*. Эти поля называются служебными. В служебных таблицах (кроме object) все поля служебные. В таблицах данных (кроме object) есть только одно служебное поле — id.

Некоторые таблицы и некоторые поля этих таблиц входят в минимальную конфигурацию системы. Такие таблицы и поля называются *врожденными*.

37. Врожденные таблицы

Таблица type (служебная)

id int	Абсолютный номер типа
id2 int default 0	Оригинал по вложенности
id22 int default 0	Оригинал по наследованию
code text not null	Код типа = имя таблицы данных
name text default ''	Имя типа (русское)
parent int default 0	Родитель по дереву вложенности

parent2 int default 0	Родитель по дереву наследования
exec boolean default false	Тип имеет функции? (выполнимый тип)
abstr boolean default false	Тип абстрактный? (у такого типа нет объектов)
abc boolean default false	Алфавитный порядок полей при показе (иначе — по порядку их добавления в тип)
icon text	Икона (если не указано, то берется из имени типа)
imadeit text default ''	Кто добавил тип
itcamefrom text default ''	Источник описания типа (для тестов — учебник)
about text default ''	Общий смысл типа
constraint **pk_type** primary key (id)	Первичный ключ
constraint **uq_type** unique (code)	Код типа не должен повторяться
constraint **fk type_id2** foreign key (id2) references type on delete cascade	Ярлык по вложенности
constraint **fk_type_id22** foreign key (id22) references type on delete cascade	Ярлык по наследованию
constraint **fk_type_parent** foreign key (parent) references type	Дерево вложенности
constraint **fk_type_parent2** foreign key (parent2) references type	Дерево наследования
constraint **ch type code** check (code REGEX '[A-z_]+[A-z_0-9]*')	Ограничения на код типа

Таблицы данных врожденные:

Таблица object

id int	Абсолютный номер объекта
code text not null	Код объекта (короткое)
name text default ''	Имя объекта (длинное, русское = **фамилия)**
parent int default 0	Родитель по дереву
type int default 0	Тип объекта (тип-центр)
id2 int default 0	Оригинал

creation_date timestamp default dateob()	Время создания объекта
last_update timestamp default dateob()	Время последнего изменения объекта
constraint **pk_object** primary key (id)	Первичный ключ
constraint **fk_object_type** foreign key (type) references type	Связь с типом
constraint **fk_object_parent** foreign key(parent)references object on delete cascade	Дерево объектов
constraint **(k_object_id2** foreign key (id2)references object on delete cascade	Ярлык
constraint **uq_object** unique (parent, code),	У одного родителя код детей не повторяется

Таблица xuser

id int	***Абсолютный номер***
pwd text not null	Пароль
hello text	Приветствие
constraint **pk_xuser** primary key (id)	***Первичный ключ***
constraint **fk_xuser** foreign key (id) references object on delete cascade	*Связь с таблицей объектов*
constraint **uqxuser** unique (pwd)	Пароли в системе не должны повторяться

Строчки, выделенные справа *жирным курсивом,* есть во всех таблицах данных, и мы опустили их (мысленно добавьте их к таблицам данных, идущим ниже), из-за чего некоторые таблицы данных оказались пустыми.

Таблица folder (Папка — пустая таблица).

Таблица **qfolder** (Папка запросов — пустая таблица).

Таблица query (Запрос)

body text	Тело запроса
pardesc text	Описание параметров
headlevel int default 0	Разрезание результата выполнения запроса на отдельные таблицы — сколько ключей проверять на постоянство

Таблица **event** (Событие)

start date DATE	Начало
end date DATE	Окончание

Таблица **person** (Человек — открытые данные)

first name text default ''	Имя
second name text default ''	Отчество
sex m Boolean	Мужчина, а не женщина

Таблица **person2** (О человеке — закрытые данные)

person int	Абсолютный номер по таблице открытых данных (не ключ) — уникален
birthday date	День рождения
addr text default ''	Адрес

Служебные таблицы:

Таблица **translator** (длинные имена полей)

tabcol text	Например **object::code**
rus text	Например: **код**
constraint **pk_translator** primary key (tabcol)	

Таблица **help** (помощь — справка)

path text	Например: **/o/myfolder/example**
help text	Например: **Это пример**
constraint **pk_help** primary key (path)	Первичный ключ

Таблица **errlog** (зафиксированные ошибки)

n int default NEXTVAL('errlog_id')	Номер по порядку появления (первичный ключ)
err text	Текст сообщения

Таблица **shadowcols** (теневые данные)

tab text	Имя таблицы Первичный ключ
col text	Имя колонки
tctype char(1)	**c** для поля в БД (**col**), **d** для документа (**DOC**), **p** для картинки (PIC)

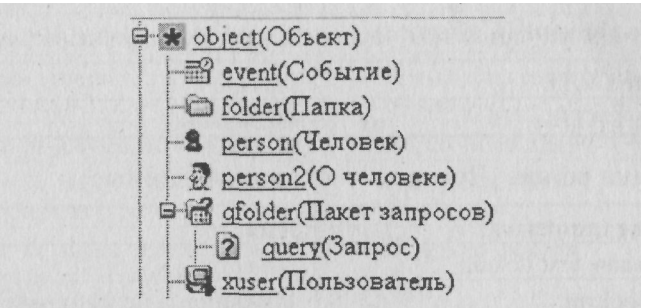

Рис. 27. Врожденные типы — вложенность

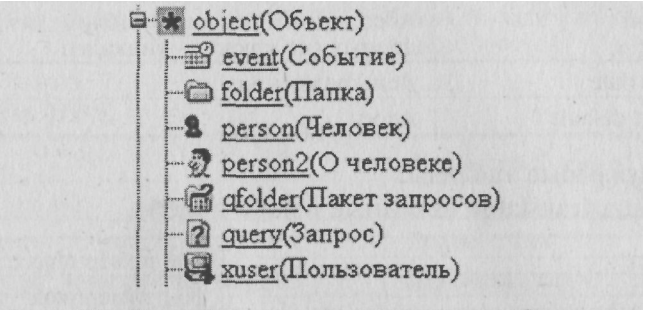

Рис. 28. Врожденные типы — наследование

38. Что уже готово

Несколько приложений, построенных на основе врожденных таблиц, широко применимы. У администратора есть возможность добавить эти подсистемы к существующей FTS-базе.

На сегодня это следующие подсистемы:

(20) process manager	Управление процессами
(30) project manager	Управление проектами, начальный блок
(31) project plus N	Управление проектами, очередной блок

Чтобы добавить любую из этих подсистем, администратор должен:

1) в дереве вложенности установить для типа object флажок function;

2) в дереве объектов вызвать список функций объекта о (корня);
3) выполнить функцию, соответствующую добавляемой подсистеме.

39. Управление процессами

Всякое производство можно представить себе как совокупность процессов. Процессы могут порождать дочерние процессы. При этом нужно иметь возможность по данному процессу собрать все его процессы-потомки.

Каждый процесс знает, кто его создал *(автор)* и какой процесс его породил (предпроцесс). Кроме того, самый первый процесса *(исток,* который не дочерний ни для какого другого процесса) фиксируется во всех процессах-потомках.

Таблица process

id int	Абсолютный номер
author INTEGER default 0	Автор
seed INTEGER default 0	Исток
father INTEGER default 0	Предпроцесс
constraint **fk_process_author** foreign key(author)references xuser	Связь с автором
constraint **fk_process_seed** foreign key(seed)references process	Связь с истоком
constraint **fk_process_father** foreign key(father)references process	Связь с предпроцессом
constraint **pk_process primary** key (id)	Первичный ключ
constraint **fkprocess** foreign key(id)references object on delete cascade	Связь с таблицей объектов

40. Управление проектами

Проект состоит из задач. *Задачи* (task,) могут содержать другие задачи. Задача имеет начало и конец по плану и по факту. Одни задачи не могут начаться раньше, чем закончились другие. Тип Задача унаследован от типа *Событие (***event***).*

Таблица **event**	
id int	Абсолютный номер
start date DATE	День начала плановый
end date DATE	День окончания плановый
constraint **pk_event** primary key (id)	Первичный ключ
constraint **fk_event** foreign key(id)references object on delete cascade	Связь с таблицей объектов
Таблица **task**	
id int	Абсолютный номер
started BOOLEAN	Началось?
ended BOOLEAN	Закончилось?
fact end DATE	Когда закончилось?
constraint **pk_task** primary key (id)	Первичный ключ
constraint **fk_task** foreign key(id)references object on delete cascade	Связь с таблицей объектов

Типы **task_xx** унаследованы от типа **task,** тип **wait_xx** хранит зависимости между задачами типа **task_xx.** Такая пара типов вводится по команде администратора «project plus N» *(Управление проектами, очередной блок;* смотри выше) и означает, что текущие задачи не будут зависеть от задач прошлых. По команде: project manager — *Управление проектами, начальный блок* — тоже вводится пара таких типов.

Далее мы не будем указывать поля **и** ограничения, общие для всех таблиц данных.

Тип **wait** xx

task INTEGER default 0	Задача, которую ждем (ключ)
constraint **fk_wait_xx_task** foreign key(task)references task xx	Связь с другими задачами

Для того чтобы указать, что задача А ждет задачу В, надо под объектом А создать объект ***зависимость,*** указывающий на задачу В.

Тип **manager** унаследован от типа **xuser.** Его специфика в том, что он (согласно дереву вложенности) может создавать задачи.

Примечание. *Задача может создавать задачу внутри себя. Точно так же (смотри выше) процесс может создавать внутри себя. Такие типы называются рекурсивными. В дереве вложенности это показано ярлыком, помещенным под оригинал.*

Тип *Задача* умеет выполнять следующие функции:

(И) bad plan	Противоречия в плане (например, плановые сроки противоречат вложенностям и зависимостям, начало позже окончания, не проставлены сроки и пр.)
(12) all tasks	Показать все задачи
(13) not finished tasks	Показать не законченные задачи
(1) weeks	Недели календаря, которые надо отобразить вперед начиная с сегодня
(2) days	Дни календаря, которые надо отобразить вперед начиная с сегодня

41. Анкетирование и тестирование

Эта подсистема добавляется через выполнение скрипта (смотри ниже) и содержит следующие типы и объекты:

testfolder	Тип умеет выполнять четыре функции (смотри ниже)
abcd	Тип для выпадающего списка вариантов
a,b,c,d	Четыре варианта (объекта) типа **abcd**

Функции **testfolder**

1	abc order:5544332211>=%	Вычисляет процент выполнения теста, выдает строчки в алфавитном порядке фамилий, позволяет установить пороги для пятибальной шкалы
2	% order:5544332211>=%	То же самое, но вывод в порядке процентов выполнения, от лучших к худшим
3	random lock	Позволяет запереть бланки тестов от несанкционированного их изменения
4	faults	Перечисляет ошибки, допущенные при сдаче теста (начиная с самых частых), и фамилии тех, кто ошибся

42. Добавление функций

Если тип, добавленный вручную, по имени совпадет с теми, что известны FTS (это может быть один из типов перечисленных выше подсистем), то он автоматически получает весь список функций, известных для этого типа. Кроме типов, описанных выше, возможны еще типы, известные системе. Например, тип father выдает всех детей вместе с временем их последнего изменения (отсортировано по кодам или по времени изменения).

Внимание! Чтобы функциями можно было пользоваться, пусть Администратор поставит в дереве вложенности для этого типа флажок function.

Если решено добавить новую функцию (списка функций), то это означает:

1) добавление нового типа (Администратор);
2) добавление нового Java-класса (Java-программист).

Разумеется, должны быть проверены следующие условия:

1. Функция не реализована ни для одного из типов, известных системе.
2. Функция не может быть реализована SQL-запросом.
3. Имя типа не известно системе.

Пример Java-класса, пригодного к добавлению, смотри в приложении *2.*

43. Публикации

Всюду, где вычисляются запросы и функции, предусмотрена возможность *публикаций.* Публикация превращает *динамически* вычисляемый файл, доступный *с паролем,* в *статический* файл, доступный *без пароля,* и сообщает адрес, по которому расположен этот файл. Пошлите этот адрес по почте тем, кому вы хотите продемонстрировать полученные данные, и они смогут их посмотреть.

Публикации хороши для периодических отчетов. В этом случае по адресу будет расположен последний опубликованный отчет.

Администратор

44. Проектирование: пример

Акульшина Инга Владиленовна любезно предоставила нам следующие таблицы для того, чтобы мы спроектировали подсистему FTS, учитывающую исследовательскую работу школьников и школьных учителей.

Карточка ученика

Имя
Отчество
Фамилия
Дата рождения
Класс
Классный руководитель
Научный руководитель
Предмет, по которому ведется исследовательская работа
Тема исследовательской работы
Даты выступлений
Участие в конференциях
Результаты предыдущего года обучения (оценки за год)
Результаты текущего обучения (оценки за триместры, полугодия)
Руководитель индивидуального маршрута
Предметы индивидуального маршрута
Продвижение по индивидуальному маршруту

Карточка учителя

Имя
Отчество
Фамилия
Дата рождения
Предмет, по которому ведется исследовательская работа
Классы, где ведется исследовательская работа

Ученики, которые ведут исследовательскую работу
Кабинет
Нагрузка на учебный год
Категория
Методический день
Методическое объединение
Методическая тема
Выступления
Публикации
Разработка индивидуального маршрута

Покажем, как преобразовать это описание данных FTS-модель (звездочкой помечены добавленные строки).

Ученика = Тип1

Имя	Поле: Текст (унаследовано)
Отчество	Поле: Текст (унаследовано)
Фамилия	Поле: Текст (унаследовано)
Дата рождения	Поле: Дата (унаследовано)
*Класс	**Тип3**
Класс	Ключ: **Тип3**
Классный руководитель	Ключ: **Тип2**
*Исследовательская работа	**Тип5** вложен в **Тип1**
Предмет, по которому ведется исследовательская работа	Вложенность объекта
Научный руководитель	Ключ: **Тип2**
*Научный руководитель2	Поле: Текст
Тема исследовательской работы	Поле: Текст
*Выступление	**Ярлык на Тип5** вложен в **Тип5**
Даты выступлений	Теневое поле: **Тип5:**
*Конференция	**Тип7**
Участие в конференциях	**Ярлык на Тип5** вложен в **Тип7**
Результаты предыдущего года обучения (оценки за год)	Поле: Документ **(тип** 8)
Результаты текущего обучения (оценки за триместры, полугодия)	Поле: Документ **(тип** 8)

*Маршрут	Тип8 вложен в **Тип1**
Руководитель индивидуального маршрута	Ключ **Тип8: Тип2**
Предмет маршрута	**Тип 10** вложен в **Тип8**
*Зачетная тема	**Тип11** вложен в **Тип10**
Сдано?	Поле Булево
Продвижение по индивидуальному маршруту	Вычисляемое

Учитель = Тип2

Имя	Поле: Текст (унаследовано)
Отчество	Поле: Текст (унаследовано)
Фамилия	Поле: Текст (унаследовано)
Дата рождения	Поле: Дата (унаследовано)
Предмет, по которому ведется исследовательская работа	Вычисляемое
Классы, в которых ведется исследовательская работа	Вычисляемое
Ученики, которые ведут исследовательскую работу	Вычисляемое
*Кабинет	**Тип12**
Кабинет	Ключ: **Тип12**
Нагрузка на учебный год	Поле: Целое число
Категория	Поле: Целое число
*Рабочий день	**Тип13**
Методический день	Ключ: **Тип13**
*Методическое объединение	**Тип14**
Методическое объединение	Ключ: **Тип14**
Методическая тема	Поле: Текст
*Исследование	**Тип 5**
Выступления	**Ярлык на Тип5** вложен в **Тип2**
Публикации	**Ярлык на Тип5** вложен в **Тип2**
Разработка индивидуального маршрута	Вычисляемое

По правому столбцу понятно, какие новые типы надо создать, какие поля и ключи добавить, какие SQL-запросы разработать.

45. Зачем нужен скрипт и как его составить

Если мы просчитали, что для нашей данной подсистемы необходимы определенные типы, определенным образом между собой связанные, то в большинстве случае, создание таких связанных между собой типов можно описать довольно простым формальным языком. Такой язык (скрипт), его основные конструкции, правила написания и применения мы и опишем ниже.

Скрипт выполняется там же и так же, как последовательность SQL-операторов при импорте (см. ниже главу «Экспорт-импорт»). Однако есть некоторые правила, которые надо соблюдать.

1. Первая строчка должна начинаться со слова SCRIPT.
2. Часть строки после // (двойной слэш) до конца строки — комментарий.
3. Заглавные и маленькие буквы различаются.
4. *Первая часть* скрипта (после первой строки со словом SCRIPT до строки со словом SET, наличие этой строки обязательно) перечисляет типы, которые надо добавить в систему — по одному типу на строку. Если в строке есть подстрока NAME, то код (короткое имя) нового типа — до подстроки, длинное имя — после. Чтобы сообщить системе, что тип запускает функции, добавьте FUN в конце строки.
5. *Вторая часть* скрипта (после единственной строки со словом SET) работает только с теми типами, которые есть в системе к концу выполнения первой части.
6. Каждая строчка второй части не зависит от других строчек второй части и выполняется сразу после ввода, прежде чем будет обработана следующая строчка или (если эта строчка последняя) завершена обработка скрипта.

Перечислим конструкции, допустимые во *второй части:*

Конструкция	*Смысл*
<t1> INTO <t2>	Тип t1 *вложить* в тип t2

Конструкция	*Смысл*
<t1> INT02 <t2>	Тень типа t1 создать и *вложить* в тип **t2**
<t1> FROM <t2>	Тип **t1** *унаследовать от* типа **t2**
<t1> FROM2 <t2>	Тень типа t1 создать и *унаследовать от* типа **t2**
<t>::<c><f>[<name>]	В тип **t** добавить поле с *формата* f (длинное имя <name>)
<t>::<c><f>2[<name>]	В тип **t** добавить поле с *формата* f (длинное имя <name>) и объявить поле *теневым*
<t1>::<c>KEYREF<t2> [NAME<name>]	В тип t1 добавить ключ с, ссылающийся на тип **t2** (длинное имя <name>)
<t1>::<c>KEYREF2 <t2> [NAME<name>]	В тип t1 добавить ключ с, ссылающийся на тип **t2,** и объявить ключ теневым (длинное имя <name>)
<t>OBJECT<path> [NAME<name>]	Создать такой объект типа t, чтобы путь к нему был **path** (длинное имя <name>)
<path>OBJECT2<path2>	Под объектом, путь к которому **path,** создать тень объекта, путь к которому **path2**
<имя типа>::<имя поля> BASE <тип ссылочной таблицы>:<CODE базового значения>	Базовое значение для ключа
<имя типа>::<имя поля> BASE <базовое значение как строка>	Базовое значение для поля
<имя типа> [CON <имя иконы>	Задать имя иконы для типа

Допустимые *форматы* перечислены в следующей таблице.

Формат	*Смысл*
REAL	Вещественное число
INT	Целое число
TEXT	Текст
DATE	Дата
TIME	Время
TIMESTAMP	Дата и время
BOOLEAN	Да — нет
PIC	Картинка
DOC	Документ

46. Общий скрипт для двух описанных примеров

```
SCRIPT
testfolder //Тестовая папка = экзамен или зачет
abcd //Варианты
workday //Рабочие дни недели
group //Группа = класс
metgroup //Методическое объединение учителей
student //Ученик
teacher //Учитель
course //Курс = Предмет маршрута = Учебник
chapter //Тема для зачета = глава учебника
portfolio //Папка для творческих работ = Маршрут
research //Творческая работа = Исследование
conference //Конференция
session //Заседание = сессия = секция
classroom //Кабинет = тематическая учебная комната
SET
folder OBJECT /o/abcd //папка для вариантов
abcd OBJECT /o/abcd/a //вариант
abcd OBJECT /o/abcd/b //вариант
abcd OBJECT /o/abcd/c //вариант
abcd OBJECT /o/abcd/d //вариант
//////////////////////
folder OBJECT /o/workdays
workday OBJECT /o/workdays/1
workday OBJECT /o/workdays/2
workday OBJECT /o/workdays/3
workday OBJECT /o/workdays/4
workday OBJECT /o/workdays/5
/o/workdays/1 NAME Понедельник
/o/workdays/2 NAME Вторник
/o/workdays/3 NAME Среда
/o/workdays/4 NAME Четверг
/o/workdays/5 NAME Пятница
//////////////////////
student INTO group
```

```
student FROM person
teacher FROM person
teacher FR0M2 xuser //Теперь учитель может войти
                    //в систему как пользователь
portfolio INTO student
portfolio INT02 teacher
course INTO portfolio
research INTO course
chapter INTO course
chapter FROM testfolder //Теперь глава может считать
                        //результаты тестирования
conference FROM event
session INTO conference
//////////////////////////////////////////////////
group::teacher KEYREF teacher //Классный руководитель
research::teacher KEYREF teacher //Руководитель
                                 //исследования из учителей
research::leader TEXT //Руководитель исследования любой
research::title TEXT //Тема исследования
research::presentation DATE2 //Дата выступления — теневое
                             //поле
portfolio::teacher KEYREF teacher //Руководитель маршрута
chapter::done BOOLEAN //Зачет сдан
teacher::classroom KEYREF classroom //Кабинет,
                                    //закрепленный за учителем
teacher::yearload INT //Годовая нагрузка учителя в часах
teacher::metday KEYREF workday //Методический день
teacher::metgroup KEYREF metgroup //Методическое
                                  //объединение
teacher::mettopic TEXT //Тема исследовательской работы
portfolio::results.xls DOC //Результаты периода (года)
                           //в таблице EXEL
```

47. Скрипт для запросов

В таблицах осталось 5 полей, которые мы отнесли к вычисляемым. Для этих полей можно составить SQL-запросы, а

можно указать структуру запроса по правилам скрипта, изложенным ниже. Во втором случае SQL-запрос будет сгенерирован системой в момент обработки этого фрагмента скрипта. Такие фрагменты строятся по следующим правилам.

1. Фрагмент расположен внутри второй части.
2. Внутри фрагмента допустимы следующие строки (те же конструкции имеют иной смысл).

Конструкция	*Смысл*
<tl> INTO <t2>	Тип **t1** соединить по вложенности с типом t2
<tl> INTO2 <t2>	Ярлыки на тип tl соединить по вложенности с типом t2
<tl>::<c>KEYREF<t2>	Тип tl соединить по ключу с типом t2
<path>QUERY<p>	Первая строка фрагмента. Будет создан объект-запрос (тип query), в тело будет записан сгенерированный запрос, в описание параметров — строка p
<t1>::<c>[,<t1>::<c>]*OF<t2>	Поле с таблицы t1, (далее по списку) будет выбрано из таблиц. Опорной таблицей будет t2. Остальные таблицы, перечисленные в этой строке (t1), будут связаны с ней через абсолютный номер (id)
SET	Последняя строка фрагмента

Примеры применения скрипта для запросов смотри в приложении 2.

Заказчик

48. FTS как объяснительный принцип

Понятия *область видимости, область ответственности* и *область компетенции* очень удобно применять при решении самых разных проблем.

Почему человек попал под машину? У него была слишком узкая *область видимости*.

Почему на границе двух участков мусорная свалка? Потому что есть территория, не попадающая ни в чью *зону ответственности*.

Почему течет водопроводный кран? Потому что у сантехника ***область компетенции*** уже, чем *область ответственности*.

49. Аналог: Cefey

http://www.cefey.ru/cmd/products/200504061154-2214.htm
ЦЕФЕЙ-ЭТАЛОН

Редактор классов

Редактор классов предназначен для создания и редактирования классов и объектов, описывающих структуру и внутренние процессы на предприятии. Управление процессом изменения информационного пространства предельно облегчается использованием:

- многооконного режима;
- закладки *стиль отображения,* регламентирующей объем отображаемой информации о классах и объектах;
- удачно подобранных значков идентификации классов;
- диалогового окна *Поиск класса/объекта;*
- приемов автоматического создания отдельных классов быстрыми операциями (запросов, экранных форм, отчетов и документов).

Окно Редактора классов является основным окном для работы в системе, из которого раскрываются окна других редакторов.

Редактор методов

Редактор методов предназначен для редактирования и отладки методов, реализующих автоматические действия и расчеты, изменя-

ющие информационную среду системы с целью решения задач управления ресурсами предприятия. Для описания и редактирования методов:

- разработан встроенный объектно-ориентированный *язык Эталон;*
- настраивается цветовое выделение синтаксических конструкций и ошибок;
- используются окна *Поиск, Замена* и *Помощник,* организующие список доступных классов, объектов и методов;
- выполнение методов отслеживается *Отладчиком,* в отдельных окнах которого:
  - можно проследить пошаговое прохождение операторов метода с остановками на точках прерывания, расставленных пользователем;
  - просмотреть динамическое изменение значений входящих переменных и определенных пользователем выражений;
  - получить стек методов, исполненных на данный момент, включая вложенные.

Редактор экранных форм

Редактор экранных форм предназначен для коррекции экрана любого назначения: независимого интерфейсного, ввода и просмотра данных справочников и балансовых счетов, ввода проводок и прочих, создаваемых в редакторе классов. Кроме того, в системе «Эталон» реализована возможность создания **составного окна,** в секциях которого пользователь по своему желанию может поместить экранные формы, отчеты, документы. При этом данные, отображаемые в секциях окна, можно согласовать, тогда, например, можно корректировать и добавлять данные в таблицу в соответствии с результатами отчета, вводить информацию в группу связных справочников и т. п.

Редактор документов

Редактор документов предназначен для просмотра и корректировки как внешнего вида, так и содержания документов различных типов:

- задавая условия выборки записей из таблицы в диалоговых окнах *Фильтр* и *Поиск;*
- группируя данные с подведением итогов по группам;
- используя библиотеку стандартных документов.

Редактор отчетов

Редактор отчетов предназначен для работы с любыми формами отчетов, созданных им в редакторе классов или хранящихся в библиотеке стандартных отчетных форм. Возможно редактирование

- как *внешнего вида* отчета:
  - изменяя ширину столбцов, высоту строк, положение столбцов;
  - замораживая столбцы;
  - сворачивая и раскрывая группы;
  - определяя цвет фона и текста;
  - переопределяя шрифт отдельных фрагментов отчета;
- так и *содержательной части* отчета:
  - добавляя и удаляя столбцы;
  - задавая группировку и наличие итоговых строк по группам;
  - находясь в отчете, корректировать данные соответствующей таблицы.

Редактор графического ввода информации справочников

В КСУ «Эталон» реализован инструментарий, позволяющий графически вводить и хранить в справочниках информацию о величинах, изменяющихся в течение времени.

Сервисные функции позволяют:

- производить **генерацию** отдельных классов или системы в целом;
- получать **граф** связей классов системы;
- управлять **правами доступа** пользователей к ресурсам системы;
- в соответствие с задачами участка учета или управления создавать **рабочие места** пользователей;
- получать **статистическую** информацию о составе классов системы;
- выполнять **импорт** данных из внешней системы;
- **архивировать** и **восстанавливать** схемы данных системы.

Компиляция классов выполняется для создания локального исполнимого кода на клиентской части системы или отдельном рабочем месте пользователя. Процесс компиляции запускается пользователем принудительно или автоматически при выполнении класса.

Генерация позволяет провести изменения в базе данных системы в соответствии с результатами проектирования информационного пространства.

Наше сравнение FTS и Cefey смотри в следующей таблице.

FTS	ЭТАЛОН-ЦЕФЕЙ
Управление правами доступа интегрировано в дерево объектов	Управление доступом — сервисная функция

FTS	ЭТАЛОН-ЦЕФЕЙ
Два дерева типов все время открыты	Один граф связей классов. Получить его — сервисная функция
Никакой специальной процедуры генерации. Всякое изменение работает на лету	Генерация классов или системы — сервисная функция

50. Аналог: TreeLogy

Эта система построена в идеологии OLAP
http://www.treelogy.ru/cd.php3?id=1

Универсальная логика организации данных:

Программа **Treelogy** оперирует с объектами.

Объектами могут быть фирмы, их подразделения, товары, деньги, сотрудники...

Объекты имеют вложенную структуру, то есть одни объекты можно перемещать в другие.

Деятельность предприятия рассматривается как совокупность перемещений рассматриваемых объектов.

Каждое перемещение сопровождается произвольным по количеству набором характеристик.

Действительно, что может происходить с объектом с течением **времени?**

Объект может:

- изменять свои **координаты** в пространстве;
- изменяться **количественно;**
- изменяться **качественно.**

Поэтому в **Treelogy** все события, имеющие отношение к деятельности предприятия, фиксируются в виде записей в таблицу, называемую таблицей перемещений.

Каждое перемещение (изменение состояния) объекта имеет следующий формат:

Стандартные поля:

- **Дата/Время** — фиксирует **время** произошедших изменений;
- **Что, Откуда, Куда** — фиксируют **изменение положения** в пространстве объектов;
- **Количество** — **количественные** изменения.

Дополнительные поля:

- **качественные** изменения — произвольные **характеристики** (создаваемые самим пользователем), соответствующие данному перемещению (изменению состояния) объекта.

Таким образом, полная первичная информация о деятельности предприятия **логически** хранится в единой таблице перемещений (колонками являются стандартные и произвольные характеристики), которую с помощью имеющихся в программе конструкторов можно интерпретировать, анализировать и строить интерфейсы клиентских мест.

Такая структура базы данных универсальна и не изменяется в зависимости от предприятия! Это дает возможность **привязать** конструкторы Запросов и Форм именно к этой структуре данных, что существенно увеличивает их функциональность, **упрощает и ускоряет** организацию управленческого учета, делает проект **открытым** для внесения любых изменений и **независимым от разработчика,** так как изменения модели управления не затрагивают структуру данных.

Наш комментарий: это *другая модель* данных.

51. IT-рынок и человеческий фактор

Прежде чем выводить на рынок новый товар, бизнесмен должен определить круг потенциальных покупателей этого товара и определить те потребности этих потенциальных покупателей, которые другими товарами не покрываются.

Кто же потенциальный покупатель FTS и какие его потребности нельзя покрыть иными товарами?

	FTS	*Другие системы*
Сотрудник предприятия	Управляет своими данными, принимает решения	Следит за одним процессом
Руководитель направления	Анализирует данные	Следит за несколькими процессами
Хозяин предприятия	Выдвигает новые идеи	Принимает решение

Как показано в таблице, одни и те же роли в FTS и в других системах требуют различной по структуре деятельности.

В своей книге «Теория уровней и модель человека» (Москва, 2005) я показываю, что это деятельности различных уровней, и привожу номера этих уровней.

Деятельность. Человек.....	*Уровень*
Следит за одним процессом	3
Следит за несколькими процессами	4
Принимает решение	5
Анализирует данные	6
Выдвигает новые идеи	7

Несколько упрощая, можно сказать, что каждый человек имеет свой любимый уровень и что он стремится проявить его на производстве. Следующая таблица показывает распределение любимых уровней в экономике России и США (обоснование — в книге «Теория уровней»).

Уровень	**Россия**	**США**
8	**1%**	**1%**
7	2%	4%
6	4%	10%
5	8%	25%
4	15%	25%
3	35%	20%
2	25%	10%
2	10%	5%

Если заметить, что другие системы не работают с уровнями выше 5, то на долю FTS остается 6% занятого населения в России и 14% в США только за счет уровней 6 и 7. Это минимальная оценка.

Знаток

52. Другие интерфейсы

Кроме основного интерфейса с тремя деревьями есть еще несколько более простых.

Не-Дерево показывает только *дерево объектов* (остальные два дерева в этом интерфейсе увидеть нельзя). В этом режиме *нельзяредактировать*данные.

Экран имеет следующий вид:

Путь до текущего объекта (можно подняться вверх)	
Дети текущего объекта (можно спуститься вниз)	Текущий объект (можно увидеть данные и выполнить функцию без параметров)

Если детей нет, левое окно исчезает, правое окно занимает всю ширину экрана.

Следующие интерфейсы применяются при анкетировании и тестировании. Для попадания в интерфейс пользователь должен ввести абсолютный номер объекта и его код. Возможно редактирование бланков и сохранение результатов.

Один бланк. Пользователь видит только поля типа, последнего в цепочке наследования.

Пакет бланков. Объект — папка, содержащая бланки. Слева линки на бланки, справа открываются сами бланки.

Следующие интерфейсы позволяют вычислить функции объекта. Надо знать номер объекта и его код. Третий параметр — код функции (появляется в скобках перед названием функции). Четвертый параметр — параметры функции. Так работает***однафункция.***

Пакет функций — получает два параметра (абсолютный номер **и** код объекта). Слева линки, кликнув, справа получаем выбор функции из списка, ввод параметра, запуск на выполнение.

Список функций. Как в предыдущем пункте, но для одного объекта.

Следующие интерфейсы предназначены для самостоятельной работы школьников (студентов) с наглядными материалами.

Дерево документов. В левой полосе показывает дерево объектов. В средней полосе показывает линкованный перечень документов выделенного в дереве объекта. В правой части открывается любой из этих документов.

Пакет документов. Как и предыдущий интерфейс, но без левой полосы (дерева).

Отдельный вход (index2.html) позволяет выйти на все перечисленные интерфейсы.

Если при входе в главный интерфейс (index.html) пометить CHECKBOX, то появится рамка внизу справа. В ней будет появляться подсказка всякий раз, когда вы наведете мышку на кнопку справа или слева.

53. Компьютерный урок

Последние два интерфейса позволяют построить, например, компьютерный урок иностранного языка для самостоятельной проработки. Опишем новый тип объектов, в который как элементы данных включим следующие документы:

1. Фрагмент кинофильма на иностранном языке.
2. Субтитры фрагмента отдельным текстовым файлом.
3. Словарь толковый.
4. Словарь картиночный.
5. Словарь обычный.
6. Вопросы для самопроверки понимания.
7. Правильные ответы для самопроверки.

Теперь у ученика есть все, чтобы по номеру и коду объекта получить перечисленные документы и проработать урок в интерфейсе «Пакет документов». Учитель, чтобы найти нужный урок и рекомендовать его ученику, пользуется интерфейсом «Дерево документов».

Заказчик

54. Экономика инноваций

Существует несколько моделей внедрения инноваций в процесс производства. На таблице — *две такие модели*. В каждой модели работают одни и те же *роли,* но *уровни* деятельности на этих ролях различны. Цифры в таблице — уровни. Подробности в моей книге «Теория уровней и модель человека».

Роль	FTS	Реформа российского образования
Авторы	7	6
Организаторы	6	4
Исполнители	5	2

Реформа образования, приняв модель 6-4-2, согласилась с тем, что для исполнителя *новая технология должна стать не новой, а старой*. Но! *По теории уровней на уровне 2 изменения в сознании невозможны*. Это означает, что все новации должны быть изучены только однажды — в вузе. Следовательно, обучение в этой системе тех, кто уже работает, — пустая трата денег.

Внедрение инноваций по модели FTS сопряжено для исполнителей с определенным риском, так как уровень 5 должен принимать самостоятельные решения. И за это исполнителям надо платить.

Администратор

55. Операции

Объект о соответствует администратору. *Корнем* дерева является *текущий пользователь* (это может быть администратор, но не обязательно). *Структура адреса* описана ранее. *Как определить адрес* конкретного узла, мы расскажем ниже.

Состояние системы: объекты

edit	Редактировать элементы данных (теневые данные редактируются построчно)	Кроме псевдоузлов
delete	Удалить узел	Кроме корня
type	Изменить тип объекта	Только оригиналы, кроме о
base	Закрыть базу данных. Для нового открытия придется перезапустить http-сервер	Только в о
shortcut	Создать ярлык и поместить его под узел с данным адресом	Кроме о
new	Создать новый объект (много объектов)	Только оригиналы
move	Передвинуть оригинал под оригинал с данным адресом	Кроме корня
find	Найти оригиналы по критерию (или найти ярлыки по критерию)	Только в о

Примечание. *В деревьях типов псевдоузлы ведут себя так же точно, как оригиналы, хотя тени (ярлыки) ведут себя не совсем так, как оригиналы. Дело в том, что псевдоузел объекта теряет операции. Этим область видимости отличается от области ответственности. Операции над типами (кроме «найти») есть только у администратора. А у него и так область видимости и область ответственности равны и покрывают все*

узлы всех деревьев. Поэтому отличие псевдоузла от оригинала теряет смысл. Все операции над псевдоузлом происходят над его оригиналом.

Ярлык имеет те же операции, что и оригинал. Но удаление ярлыка не ведет к удалению оригинала. Удаление оригинала ведет к удалению всех его ярлыков. В остальном все операции (кроме «передвинуть») происходят над оригиналом, хотя мы работаем над его ярлыком.

Состояние системы: **вложенность типов.** Только для администратора.

sql	Подготовить запросы на создание объектов данного типа (для последующей записи в файл) без учета наследования	
new	Создать тип вложенный	
edit	Отредактировать внешние имена полей	
icon	Подобрать икону	
delete	Уничтожить тип	Кроме **object**
add	Добавить поле	Кроме **object**
base	Установить базовые значения	Кроме **object**
password	Изменить пароль администратора	Только в **object**
move	Передвинуть под другой узел дерева	Кроме корня
shortcut	Создать ярлык и поместить его под узел с данным адресом	Кроме **object**
function	Есть функции?	
name	Изменить полное имя типа	

Состояние системы: наследование типов. Только для администратора.

find	Найти тип-оригинал по критерию (или найти ярлыки типа по критерию)	Только в **object.** *Есть у всех пользователей*

sql	Подготовить запросы на *создание типа* (для последующей записи в файл) без учета наследования	
abc	Выдавать поля в алфавитном порядке внутренних имен (а не в порядке их добавления в тип)	
move	Передвинуть под узел дерева с данным адресом	Кроме корня
shortcut	Создать ярлык и поместить его под узел с данным адресом	Кроме **object**
name	Изменить полное имя типа	
icon	Изменить икону	
new	Создать абстрактный тип унаследованный	
function	Показать все функции для всех типов	Только в **object**

56. Как узнать адрес узла

Адрес узла состоит из одной буквы (указывающей на тип дерева) и целого числа. Для объектов, если первая буква маленькая, то число указывает текущий номер узла в дереве. Этот номер может меняться при каждом обновлении дерева. В остальных случаях большая и маленькая буква не различаются.

Буква	Дерево
a	Объекты, текущий номер
A	Объекты, абсолютный номер
T, t	Вложенность типов, абсолютный номер
H, h	Наследование типов, абсолютный номер
N, n	Дерево для изменения тип-множества и/или тип-центра
R, r	Дерево для автоматической генерации запросов (отчетов)

Чтобы узнать адрес линкованного узла, поставьте курсор мыши на его имя. Внизу в браузере (статус-бар) появится длинный адрес с параметрами (линк). Последний параметр после последнего знака = есть адрес данного узла. Чтобы узнать абсолютный адрес объекта, откройте его (справа) и поставьте курсор мыши на кнопки *И* или В (справа вверху), отвечающие за перенос корня дерева. Внизу в статус-баре появится длинный адрес с параметрами (линк). Последний параметр после последнего знака = есть абсолютный адрес объекта.

57. Экспорт — импорт

Иногда появляется необходимость перенести часть данных из одной FTS-базы в другую. Это может быть другая подсистема на том же сервере, а может быть подсистема на другом сервере.

Пример первый. Перенести тип — тест по физике из одной школы в другую.

Пример второй. Перенести тип и список (например, список методических объединений учителей города).

Чтобы решить проблемы такого рода, воспользуемся операциями **sql**, которые есть у нас в дереве вложенности типов и в дереве наследования типов.

Действие	Дерево	Подсистема
Получить запросы на создание типа и записать их в файл	Наследования	Старая
Получить запросы на создание объектов данного типа и записать их в файл	Вложенности, любой тип кроме **object**	Старая
Прочитать запросы из файла и выполнить их	Вложенности, тип **object**	Новая

Внимание!

- Запросы создаются без учета цепочки наследования. Это имеет смысл, так как в разных подсистемах могут быть разные цепочки и старой подсистеме про новую подсис-

тему это не известно. Если вам известно о цепочках новой подсистемы, соответствующие запросы в файл нетрудно добавить вручную.

- Для тестов *(пример первый)* и стандартных списков *(пример второй)* цепочка наследования не содержит ничего кроме начала и конца. Поэтому в этих двух случаях предыдущее замечание силы не имеет.

58. Базовые значения, составитель, источник

Для того чтобы система могла сама подсчитать, сколько заданий выполнено правильно, надо задать ей правильные ответы или, что то же самое, *базовые значения*. Система хранит эти значения в нулевой записи каждой таблицы. (Вот почему у нулевого объекта нельзя расширить тип-множество.)

Чтобы установить базовые значения, воспользуйтесь командой **base** дерева вложенности.

Внимание! *Если базовое значение не установлено, то тест не будет включен в вопрос! Поэтому, даже если вы собираетесь оценивать результаты на глаз и сравнение с базовым значением вас не интересует, какие-то значения вам придется ввести.*

Примечание. *Обычно мы просим того, кто создал тест, самому сдать экзамен. Понятно, что результат в этом случае должен быть 100%.*

Имя того, кто ввел тест, хранится в специальном поле **type::imadeit**. Русское имя поля: **составитель.** Поле **type::itcamefrom** хранит источник теста (например, название учебника). Русское имя поля: **источник.** Эти поля можно увидеть справа внизу в дереве вложенности типов. Их значения можно изменить (буква E справа от поля).

Экспорт базовых значений: используйте кнопку EXPORT в дереве вложенности справа.

59. Безопасность при тестировании студентов

Поскольку часть экзаменуемых хочет получить хороший балл обходными путями, в системе предусмотрен ряд мер профилактики.

Первая самая простая профилактика состоит в том, что правильные ответы не посылаются в браузер клиента, а есть только на сервере. Некоторые студенты пытаются узнать их, вскрывая код, присланной в браузер страницы.

Некоторые студенты пытаются зайти в бланк отличника, уже сдавшего тест, и скопировать себе его результаты* Осложнить этот ход можно следующим образом. Дело в том, что попасть в бланк можно, если задать внутренний номер объекта и его код. При создании теста эти два параметра совпадают. Однако после того, как тест сдан, у учителя есть возможность «запереть старые бланки». Для этого надо использовать функцию (3) lock папки типа **testfolder**. При *этом к коду объекта, лежащего в папке, присоединяется длинное случайное число.* Для просмотра бланка из дерева это не имеет значения. Но для входа в бланк студента, который не знает всего этого случайного числа, препятствие непреодолимое.

Некоторые студенты пытаются узнать у своих товарищей правильные ответы на вопросы. Чтобы помешать им, пользуйтесь вопросами с выбором из набора вариантов. Если вы при описании типа дадите ключу имя вида q <номер>, то система будет перемешивать варианты случайным образом при каждой подаче. Это означает, что подсказка «в первом вопросе правильный ответ А, во втором — С» не спасает. Даже повторное открытие того же самого бланка приводит к перемешиванию вариантов.

Чтобы система НЕ перемешивала, при создании теста используйте для имени поля другую первую букву (кроме q).

Ввод вопросов с вариантами в качестве длинного имени поля происходил так часто, что пришлось разработать специальный интерфейс, в котором сам вопрос и каждый вариант ответа занимают отдельное окно формы.

60. Как работает поиск

В системе есть возможность найти узел, о котором кое-что известно. Для этого воспользуйтесь командой **find.** Причем поиск в дереве объектов и в дереве вложенности может провести только администратор, поиск в дереве наследования — любой пользователь.

Система формирует запрос, соединяя постоянную часть (SELECT id FROM object WHERE) и ту часть, которую мы ей сообщим в качестве параметра, например (code = 'myfolder') или (name LIKE 'my%'). О понимании второго примера спросите у SQL-программиста.

Внимание! *Такой запрос находит только оригиналы. Если нужно искать ярлыки, то поставьте V в CHECKBOX. Учтите, что постоянная часть в этом случае более сложная и вам придется перед полем писать о в качестве короткого имени нужной таблицы, например (o.code = 'myfolder'). Те же правила действуют и в других деревьях.*

Для найденных узлов будут показаны пути. Каждый путь заканчивается кнопкой H, которая позволяет *сделать данный узел новым корнем* дерева слева.

61. Изменение одного элемента

Операция **edit,** которая позволяет изменить *элементы данных объекта, названия полей типа* и *базовые значения типа,* работает сразу со всем списком. Чтобы изменить один элемент списка, используйте *линкованную букву* рядом со значением элемента: B для базовых значений, E для элемента объектов и типов.

Внимание! *Для поля есть возможность объявить его UNIQUE так, чтобы неповторяемость значений отслеживалась средствами СУБД. Выйти на этот флажок можно через редактирование одного поля в дереве вложенности (E).*

62. Формат элемента данных

В деревьях типов можно увидеть формат элемента. При этом возможны варианты:

Формат	Что это означает
FILE или DOCUMENT	Документ
PICTURE	Картинка
Другие БОЛЬШИЕ БУКВЫ: INTEGER, LONGVARCHAR, BOOLEAN, REAL, DATETIME, DATE, TIME	Поле, хранимое в базе данных, показан его формат
Маленькие буквы, например v_abcd	Ключ, хранимый в базе данных, показана таблица (тип), на которую указывает этот ключ

63. Файлы, картинки, документы

В самом начале, очерчивая контуры будущей FTS, мы заявили, что наша система будет поддерживать любые форматы данных, для которых есть вьюер. При этом пришлось признать, что те элементы, форматы которых не поддерживает СУБД, будут храниться в файловой системе сервера.

Опыт эксплуатации первых версий системы привел нас к мысли *разделить файлы на две категории.*

Первую категорию составляют файлы, которые при своем просмотре занимают на экране не слишком много места и не мешают одновременному показу других элементов. Например, если отдел кадров составляет Личную карточку сотрудника, то небольшую фотографию лучше показывать вместе с остальными данными, на том же экране.

Вторую категорию составляют файлы, которые лучше показывать в отдельном окне. Например, трудовую биографию того же сотрудника. *За второй категорией* мы закрепим термин *«документы», первую категорию* мы будем обозначать словом *«картинки».*

Для *картинок* возможны любые форматы, понятные браузеру. Если браузер этот формат не понимает, лучше объявить элемент *документом,* а на клиенте установить вьюер. Добавление поля происходит раздельно по четырем категориям:

1. Поля.
2. Ключи.
3. Документы.
4. Картинки.

64. Большие списки

При редактировании документа в полях типа *ключ* система предлагает пользователю выбрать один из вариантов в выпадающем списке. Если список слишком велик, то возникают проблемы, во-первых, при наполнении списка на сервере (время возрастает), во-вторых, при пересылке документа пользователю (время возрастает), в-третьих, в момент выбора (выбирать неудобно). Поэтому система различает три варианта. Граница между вариантами проходит по двум пороговым числовым значениям. По умолчанию это P1 = 10 и P2=100, но администратор может изменить эти значения. Для этого в дереве вложенности типов, в корне, он должен использовать кнопку LIST. Значения разделены знаком «;» (точка с запятой). Итак, устанавливая значение ключа, пользователь должен выбрать один вариант из списка.

Если в списке значений не более P1 вариантов, то пользователю выпадающий список вариантов будет предъявлен полностью (рис. 29).

Если значений более P1, но менее P2, то в списке появится только P1 значений, после чего появится строчка MORE («еще») (рис. 30).

Если выбрать эту строчку, то откроется окно с вариантами, причем варианты, уже попавшие в список, не появятся. Слева, против каждого варианта — радиокнопка (рис. 31). Если ее выбрать, то окно автоматически закроется, а выбранный вариант добавится в выпадающий список. После чего задача сводится к предыдущей.

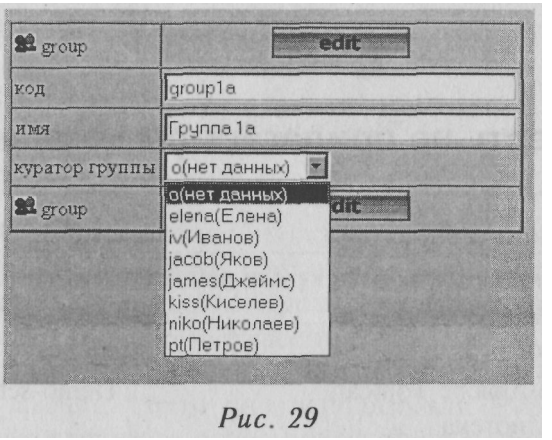

Рис. 29

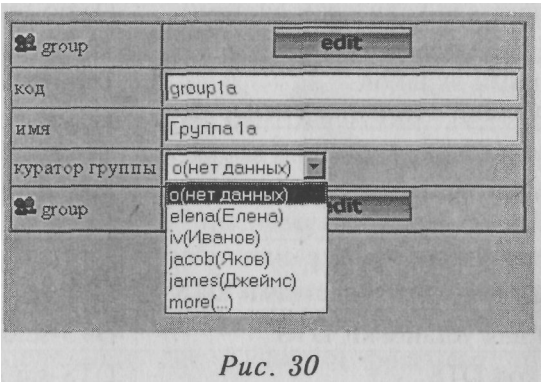

Рис. 30

Если вариантов более P2, то вместо списка появится окно для ввода текста. Считается, что пользователь знает абсолютные номера выбираемых объектов и введет один такой номер в это окно.

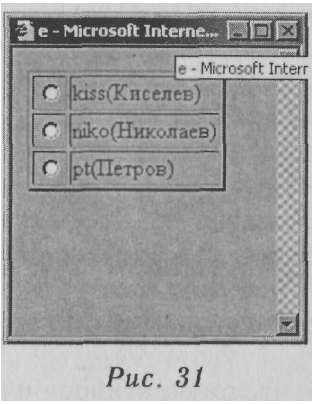

Рис. 31

Специалист

65. Что есть на прилагаемом компакт-диске

Содержание	Папка (файл)
Java, минимальный комплект для работы всех остальных систем	Java-basics
СУБД mckoi	Mckoi
http-сервер Apache Tomcat, upload-библиотека	Demo-setup
FTS, комплект для установки системы (демоверсия)	Demo-setup/feldman
Сайт с копиями экранов, иллюстрирующий основные сценарии	Tutorial
Инструкция по использованию диска	Readme.doc
Java, комплект для разработчика Java-функций	Java-more
Описания: FTS, DTS, ТУАИ, упакованные как сайт средствами DTS	DTS-sites
Материалы для установки DTS	DTS-setup
Открытый код DTS	DTS-src

Приложение 1. FTS для ленивых

Введение

В основной части книги автор старался рассказать о системе FTS как можно подробнее. При этом неявно предполагается, что читатель, независимо от его роли, энергичен, самостоятелен и любознателен. Но кроме таких читателей есть и другие. Они ленивы, нелюбопытны и зависимы от начальства. Однако если начальство захочет, чтобы они использовали FTS, невозможность для них прочитать основную часть по причине слабости их характера не должна служить препятствием для вовлечения в FTS. Вот почему специально для них написано это приложение.

Невозможность прочитать основную часть не обязательно связана с плохими чертами характера, как теми, которые перечислены выше, так и с иными. Есть значительное число людей, положительных во всех отношениях, но со складом ума не таким, как у автора этой книги. Прежде всего я имею в виду тех, чей мозг хранит информацию в картинках (а не в текстах и схемах), и кто смыслам предпочитает образы и сценарии. Этим положительным персонажам я тоже предлагаю начать с приложения.

Как и в основной части, каждый фрагмент, предназначенный для читателя, пребывающего в определенной роли, будет предваряться соответствующим заголовком. Однако, в отличие от первого учебника, для каждой роли здесь будет только один фрагмент и внутри фрагмента будет изложено все необходимое знание, так что читатель будет избавлен от необходимости читать другие фрагменты книги. При этом некоторые вещи придется повторить, но такова цена.

Читатель первого типа может и здесь найти для себя кое-что интересное. Во всяком случае, само наличие этого приложения вместе с нашей убежденностью, что оно очень важно для практической судьбы FTS, может оказаться знанием неожиданным и полезным.

Для тех, кто прочитал мою «Теорию уровней и модель человека», скажу, что основная часть предполагает логичность и по-

нимание, любовь к оригинальным идеям и интуицию, то есть уровни 6, 7 и 8. Приложение предполагает самостоятельность, исполнительность и умение работать в ансамбле, способность доводить начатое дело до конца, то есть уровни 5, 4 и 3.

Заказчик

66. Предположим, что...

Вы — руководитель предприятия (завода, школы, больницы). Вы не просто хотите компьютерно отчитаться перед внешними партнерами (налоговой инспекцией, министерством, страховой компанией), но еще использовать компьютеры для удовлетворения своего *производственно-профессионального любопытства*. При этом вы человек импульсивный. Сегодня хотите знать одно, завтра второе, послезавтра третье. И вы не терпите лжи, просто органически ее не перевариваете. Кроме того, вы человек бережливый и не станете платить сторонней фирме большие деньги неизвестно за что. Не станете вы и держать целый отдел нахлебников-программистов.

В этом случае — FTS — ***ваш*** *выбор*.

67. Найдите одного человека

...на которого вы можете положиться *абсолютно*. Назовем его Администратор. Он должен быть самостоятелен, абсолютно честен и немедленно реагировать на все важные относящиеся к делу события, прежде всего на недовольство пользователей, на их предложения-пожелания, на их (и ваши) идеи. Чтобы не рисковать всем, советую вам во время отпуска этого человека закрывать предприятие. Две недели летом, две недели зимой. Зато никакого риска.

Кроме того, этот человек должен не бояться компьютера, с радостью учиться, без колебаний обращаться за помощью к консультантам. Кроме того — и с этого надо было начать —

этот человек без ошибок переписывает длинные и непривычные слова, как кириллицей, так и латиницей, и не боится проверить английское слово по словарю.

Этот человек по своему образованию может быть программистом, но это не обязательно. Если он не программист, то у вас есть выбор: передать программистскую поддержку нам или найти себе одного или двух программистов, которые будут у нас учиться. Если Администратор по образованию программист, то, во втором варианте, учиться у нас FTS придется ему. В любом случае, *доверять профессионализму* ваших программистов вы сможете *только после их аттестации у нас*.

Не пугайтесь: программистская поддержка FTS не имеет ничего общего с поддержкой других систем, вроде SAP или 1С. Развитие системы FTS осуществляет администратор, а программисты лишь изредка выполняют его заказы, достаточно мелкие, чтобы справиться с ними за день или два. Однако если от глупого пользователя система надежно защищена, то *от глупого программиста* (и тем более *администратора) защиты нет!*

68. Установите с нами контакт

Мы научим Администратора, как ставить систему, как ее развивать, как ставить задачу программистам, как обучать пользователей. Мы будем отвечать ему на все вопросы быстро и понятно. Ваша корпоративная информация останется закрытой для нас. Администратору достаточно будет сделать демокопию вашей базы, где будут фиктивные данные. Мы получим доступ к этой базе по Интернету и продемонстрируем Администратору, что ему надо делать. После чего он, потренировавшись на демобазе под нашим контролем, сможет без труда сам продвинуть развитие реальной базы.

69. Вы получите

Единое информационное пространство, полный контроль (здесь и сейчас) над всем, что происходит в вашей компании, глубинный анализ всех процессов, достоверную, полную, не-

противоречивую, актуальную информацию в удобном для вас виде — и все это очень быстро и за очень маленькие деньги!

70. Что же делать?

Связаться с нами и как можно скорее, поведав свои мысли, проблемы и обстоятельства:

mailto: fel2@onego.ru
Фельдман Яков Адольфович.

Администратор

71. Предположим, что...

Вас назначили Администратором FTS со всеми вытекающими отсюда последствиями. Теперь вы единолично головой отвечаете за все данные системы, за ее бесперебойное функционирование и развитие. Вам придется обучать пользователей, реагировать без промедления на их жалобы, предложения, пожелания, идеи. Вам придется ставить задачи программистам (нашим или вашим — не важно). Вам придется разрешать конфликты, как содержательные, так и формальные. Но не отчаивайтесь: с нашей помощью вы непременно справитесь со всем этим клубком проблем. А восхищение вашего начальника и признательность благодарных пользователей сторицей вознаградит большой расход ваших эмоций и интеллекта.

72. Определите роли

Тех, кто будет с вами работать. Это лучше сделать *еще до* установки системы и решения всех технических вопросов с этим связанных. Если вам не удастся *определить систему ролей,* все дальнейшее не будет иметь смысл.

Если ваше предприятие — ***завод,*** то (примерный, заведомо не полный) набор ролей может выглядеть так:

- Руководитель высшего уровня;
- Руководитель среднего и младшего уровня;
- Специалисты по направлениям:

1) бухгалтер;
2) кадровик;
3) мастер цеха;
4) конструктор;
5) технолог;
6) контролер качества;
7) маркетолог;
8) пожарник;
9) охранник;
10) ...

Если ваше предприятие — *поликлиника,* то примерный набор ролей может выглядеть так:

- руководитель;
- врач;
- медсестра;
- бухгалтер;
- завхоз.

Если ваше предприятие — *школа,* то примерный набор ролей может выглядеть так:

- директор;
- завуч по направлению (несколько ролей);
- учитель по предмету (много ролей);
- классный руководитель;
- психолог;
- социолог;
- врач;
- завхоз.

73. Установка и тренировка

Первоначальную установку системы должен провести программист (ваш или наш — не важно). Первая установка покажет вам демо-базу. После этого вы сможете обратиться к учебнику на диске и потренироваться на этой базе. В учебнике показаны все операции, которые можно проделать на этой базе и копии возникающих при этом экранов.

Когда вы пройдете этот этап, и освоите основные операции, вам придется поставить другую базу и начать работу уже с ней.

74. Обучение персонала

Именно вам придется обучать всех участников процесса. Если вы работаете в школе, то для таких ролей далее есть специальный фрагмент. Прочитайте его сами и дайте прочитать исполнителям. Покажите им учебник на диске, научите им пользоваться.

75. Сохранение и восстановление данных, замена версий

Эти обязанности тоже ваши. Пока вам нужно просто о них помнить. Прямые инструкции — в прямом контакте.

Сотрудники школы

76. Другая школа

Мы считаем сегодняшний российский вариант школы принципиально вредным для всех: для детей, для учителей, для страны в целом. Мы предложили свою модель школы, которую описали в книге «Теория уровней и модель человека». В этой модели шесть направлений (более подробно о проектах Школа-2010 и Школа-2050 смотри в книге «Педагогика Луны и Педагогика Солнца», выходящей следом за «Теорией уровней» в том же издательстве):

1) компьютерное тестирование (на основе FTS);
2) компьютерный учебник (на основе FTS);
3) индивидуальность ученика (на основе FTS);
4) психотехнические игры;
5) влияние пространства;
6) управление школой (на основе FTS).

Как видно из сказанного, FTS играет в новой модели важную роль. Сам вход в новую модель через создание тестов является очень важным, и мы опишем его подробно.

77. Абсолютное тестирование

Мы ведем речь только о средней школе, оставляя в стороне младшие классы. Мы начинаем внедрение тестов со старших классов (9—10), где средний ученик уже не боится компьютера, и постепенно снижаем возраст.

В идеале все предметы всех классов должны быть разбиты на темы. Для каждой темы должны быть составлены задания на первые три уровня освоения материала:

1) узнавание;
2) различение-разграничение;
3) упорядочение, линейное выполнение.

Эти задания должны быть оформлены как тесты и погружены в FTS. Мы рекомендуем оформлять каждое задание как выбор одного из четырех вариантов.

78. Подготовка и проведение тестов

Итак, учитель готовит тесты по конкретной теме.

Администратор вместе с учителем погружает эти тесты в систему.

Учитель размножает бланки тестов внутри FTS сообразно спискам учеников (о списках будет сказано ниже).

Учитель формирует и печатает таблицу сдающих с кодами бланков.

Ученики приходят в класс, садятся у компьютеров, открывают браузеры, входят по «известному адресу», набирают код бланка (дважды) и начинают сдавать тест. Бланк, заполненный даже частично, рекомендуется сохранить. В пределах отведенного времени сохранять бланк можно многократно.

Учитель в любой момент может проанализировать тест. Результаты анализа он может сообщить ученикам или распечатать. Существует три вида анализа.

- Проценты выполнения: фамилии по алфавиту.
- Проценты выполнения: от лучших к худшим.
- Допущенные ошибки: от частых к редким.

После проведения тестов учитель, по своему усмотрению, может «запереть бланки», закрыв к ним несанкционированный доступ других студентов.

79. Списки учеников

Списки учеников формирует классный руководитель (иногда — завуч). Ученики — это объекты FTS. Учитель-предметник получает не сами эти объекты, а ярлыки на них. Таким образом, можно включать одного и того же ученика в разные группы, добавляя ярлыки на его объект неограниченно.

80. Контроль процесса

Директор или завуч в дереве объектов FTS оказываются выше учителей. Это означает, что они могут посмотреть любой тест, проведенный учениками этого учителя.

81. Компьютерный урок

Учитель иностранного языка может с помощью FTS построить уроки для самостоятельного изучения. В урок можно включить следующие элементы:

8) фрагмент кинофильма на иностранном языке;
9) субтитры фрагмента отдельным текстовым файлом;
10) словарь толковый;
11) словарь картиночный;
12) словарь обычный;
13) вопросы для самопроверки понимания;
14) правильные ответы для самопроверки.

Интерфейс ученика показывает один урок. Интерфейс учителя позволяет выбрать нужный урок.

Системный программист

Первоначальная установка системы и поддержка всей среды ее функционирования невозможны без системного программиста. Кто-то должен отвечать за функционирование локальной сети, за настройку операционных систем, за адреса машин, за активность сервера. Системный программист должен научить администратора запускать http-сервер и останавливать его вновь. Это необходимо при архивировании (восстановлении) базы и при замене версии FTS. Этим операциям системный программист тоже должен научить администратора.

82. Демоверсия

Демонстрационная версия отличается от рабочей ограничениями, препятствующими использованию версии для управления работы реальным предприятием. На демоверсии можно посмотреть основные интерфейсы и операции. Копии экранов есть на диске (demo-base). Установка демоверсии описана на диске в файле *readme.doc*.

После того как администратор научится работать с демоверсией и вы решитесь на использование FTS, свяжитесь с нами:

mailto: fel2@onego.ru

Фельдман Яков Адольфович.

Приложение 2. Тексты программ

83. Добавление Java-класса

Следующий пример показывает, как должен быть написан (и куда помещен) добавляемый Java-класс.

```
/*
* Father.java
* Created on 16 Май 2005 г., 8:35
*/
package j5feldman.proc;
import j5feldman.*;
import java.util.*;
import java.sql.*;
import j5feldman.proc.basement.Proc;
import j5feldman.proc.basement.IProc;
/**
* ©author Jacob Feldman
*/
public class Father extends Proc implements IProc{
   boolean cc;
   /** Creates a new instance of Father */
   public Father() {
   }
   public void init(XPump pp, int id)throws Exception{
      super.init(pp,id);
      map.put("71","last update order");
      map.put("72","code order");
    }
    final static String d1 = "Дети — в порядке создания";
    final static String d2 = "Дети — в порядке последнего
    обновления";
public String parDesc(int id,int k){
   switch(k){
      case 71:return d1;
      case 72:return d2;
```

```
    }
    return "????";
}
public String result(String fun,String par) throws Exception {
   this.par = par;
   char c — fun.charAt(O);
   if(c!='d'&&c!='c')return "";//throw new Exception("Wrong
   params");
   cc = c=='c';
   String s = "";
   List<String> ar = getArrayl(id);
   ar.addAll(getArray2(id));
   for(String p:ar){
      s+=p;
   }
   return
         "<table border=2><tr><td>тип</td>" +
         "<td>№</td><td>код</td><td>имя</td><td>изменен
          </td></tr>"+s+"</table><hr>";
}
public List<String> getArray(String q)throws Exception{
   List<String> ar = new ArrayList<String>();
   ResultSet rs=null;
   String code="",name="",last="",type="",s="";
   int id;
   try{
      rs = pp.select(q);
      while(rs.next()){
         type = rs.getString("type");
         id = rs.getInt("id");
         code = rs.getString("code");
         name — rs.getString("name");
         last = rs.getString("last_update");
         s="<tr><td>" +type+
               "</td><td>" +id+
               "</td><td>" +code+
               "</td><td>" +name+
               "</td><td>"+last+
```

```
                "</td></tr>";
        ar.add(s);
      }
}finally{if(rs!=null)rs.close();}
   return ar;
}
public List<String> getArrayl(int parent)throws Exception
{
   String q = "select 'o' type,o.id,o.code,o.name,o.last_update from
         object o where " +
         " o.id2=0 and o.parent ="+parent+
   " order by " +
        (cc?" o.code ":" o.last_update desc ");
   returngetArray(q);
}
public List<String> getArray2(int parent)throws Exception
{
   String q = "select '&' type,o.id,o.code,o.name,o.last_update from "+
   " object o,object o2 where " +
         " o2.id2=o.id and o2.id2>0 and o2.parent ="+parent+
   " order by " +
          (cc?" o.code ":" o.last_update desc ");
    return getArray(q);
  }
}
```

84. Скрипты — запросы для вычисляемых полей

1	Классы, в которых ведется исследовательская работа	Перечислить группы, учеников в них, портфолио в учениках, предмет в портфолио, исследования в предмете	student INTO group portfolio INTO student course INTO portfolio research INTO course object::code OF group object::name OF student object::name OF research

2	Ученики, которые ведут исследовательскую работу	Как (1), но без групп	portfolio INTO student course INTO portfolio research INTO course object::name OF student object::name OF research
3	Предмет, по которому ведется исследовательская работа	Как (2), но без учеников,	research INTO course object::name OF course object::name OF research
4	Разработка индивидуального маршрута	Перечислить группы, учеников в них, портфолио в учениках, предметы в портфолио, главы в предметах	student INTO group portfolio INTO student course INTO portfolio chapter INTO course object::code OF group object::name OF student object::name OF course object::name, chapter::done OF chapter

Рассмотрим подробно первый пример.
Смысл вычисляемого поля и смысл запроса:

Классы, в которых ведется исследовательская работа	Перечислить группы, учеников в них, портфолио в учениках, предмет в портфолио, исследования в предмете

Текст скрипта

```
SCRIPT
SET
qfolder OBJECT /o/qfx
/o/qfx/1 QUERY no
student INTO group
portfolio INTO student
course INTO portfolio
research INTO course
object::code OF group
object::name OF student
object::name OF research
SET
```

Генерируемое тело запроса

```
SELECT t0t12.code, t0t14.name, t0t19.name
FROM object t0t12, object t0t14, object t0t16, object t0t18, object
t0t19, group t12, student t14, research t19
WHERE t12.id>0 and t12.id=t0t12.id and t14.id>0 and
t14.id=t0t14.id and t19.id>0 and t19.id=t0t19.id and
t0t14.parent=t0t12.id and tOt 18.parent=t0t14.id and
t0t16.parent=t0t18.id and t0t19.parent=t0t16.id
ORDER BY t0t12.code, t0t14.name, t0t19.name
```

Содержание

Серия «Библиотека инженера»

Яков Адольфович Фельдман

Создаем информационные системы

Ответственный за выпуск
В. Митин
Макет и верстка
А. **Иванова**
Обложка
Е. Жбанов

ООО«СОЛОН-ПРЕСС»
123242, г. Москва, а/я 20
Телефоны:
(495) 254-44-10, (495) 252-36-96, (495) 252-25-21

По вопросам приобретения обращаться:
ООО «АЛЬЯНС-КНИГА КТК»
Тел: (495) 258-91-94, 258-91-95, **www.abook.ru**

ООО «СОЛОН-ПРЕСС»
103050, г. Москва, Дегтярный пер., д. 5, стр. 2
Формат 60x88/16. Объем 7,5 п. л. Тираж 1000 экз.

Отпечатано в ООО «Арт-диал»
143983, МО, г. Железнодорожный, ул. Керамическая, д. 3
Заказ № **161**