

Jet

INFO

**МАТЕРИАЛ
НОМЕРА**

**Системы управления
базами данных -
коротко о главном**

Часть 3

А ТАКЖЕ:

- **НОВОСТИ INTERNET**
- **ТЕЗКИ**
- **АНКЕТА ЧИТАТЕЛЯ**

5
1995

И Н Ф О Р М А Ц И О Н Н Ы Й Б Ю Л Л Е Т Е Н Ь



Ведущий рубрики — Александр Гагин. ЕЩЕ РАЗ ПРО ДАУНСАЙЗИНГ

Пришло время даунсайзинга Солнечной системы. Меркурий: кто за? Кто против?...

ПРИМЕЧАНИЕ.

Эта пенка некоторое время плавала по Интернету. Хотя материал и не вполне соответствует направленности нашей рубрики "Новости Internet", он показался нам слишком соблазнительным, чтобы пройти мимо.

Вашингтон, округ Колумбия. Подкомитет Финансовой Палаты по надзору за деятельностью НАСА предпринял очередную попытку урезать бюджет НАСА, приняв сегодня резолюцию о даунсайзинге Солнечной системы.

По утверждению некоего конгрессмена, пожелавшего остаться неназванным, члены Палаты от республиканской партии полагают, что Солнечная система обладает чрезмерной избыточностью, и наш долг — рационализировать планетарный комплекс, насчитывающий уже более 4.5 миллиардов лет.

В результате подобной акции сократится число пунктов, достигаемых для НАСА, что позволит этой организации осуществлять космические исследования в рамках того финансирования, которое было предложено Конгрессом ранее этим летом.

ПЛАНЕТАРНАЯ ЧИСТКА

"Итак, мы имеем три планеты земной группы", — сказал конгрессмен Проч С.Дороуги (респ., штат Делавэр), — и только одна из них реально работает! Так почему бы нам не избавиться от остальных двух, расчистив тем самым ближайшие окрестности?"

Большинство членов подкомитета, соглашаясь с настоятельной необходимостью даунсайзинга, полагают все же, что устранение сразу обеих планет — Марса и Венеры — чересчур жесткая мера.

"По Марсу у нас слишком много международных обязательств, — заявил Росс О.Ненавистер (респ., штат Калифорния), — Поэтому я думаю, что Марс при-

дется оставить, а Венеру убрать. Во-первых, жить там все равно невозможно из-за жары, во-вторых, либеральные демократы постоянно используют ее как пример того, к чему может привести глобальное потепление. Таким образом, ни с политической, ни с практической точки зрения Венера нам абсолютно не нужна."

Планета Меркурий также оказалась под угрозой уничтожения. Меркурию не хватает поддержки из-за его малых размеров и слабой видимости с Земли.

"Кому он нужен? — спрашивает конгрессмен Три Тон (респ., штат Северная Каролина), — Вы когда-нибудь видели его? Я — нет. Тогда какой от него прок? Нам не нужны бесполезные планеты. И, раз уж речь зашла о бесполезных планетах, нельзя не сказать об астероидах. Если вы видели хоть один из них, считайте, что вы видели их все. Поэтому я заявляю, что мы должны избавиться от них раз и навсегда."

"НЕТ" НЕПТУНУ

Рекомендации по даунсайзингу Солнечной системы не ограничились лишь планетами земной группы. Резолюция содержит также призывы к устранению некоторых газовых гигантов — на которые, кстати, приходится большая часть общей планетарной массы.

Большинство членов подкомитета склоняются к тому, чтобы оставить Юпитер и Сатурн, отказавшись от Урана и Нептуна.

"Юпитер содержит наибольшее число молекул, а Сатурн окружен этакими прелестными колечками, которые всем так нравятся, — сказал Респ Кон Грессмэн (респ., штат Флорида), — Уран же, напротив, совершенно неинтересен, и кольца у него какие-то грязные. Ну а Нептун, слава Богу, уж слишком далеко..."

Однако влиятельный Дайлек О.Пойдеш, конгрессмен от штата Южная Каролина, публично заявил, что он будет бороться и за исключение Сатурна. Его, в частности, раздражает успех НАСА, сумевшего отстоять полет к Сатурну в рамках проекта Кассини, что, по его мнению, является разбазариванием средств налогоплательщиков,

Конгрессмен выразил также озабоченность по поводу снующих туда-сюда летательных аппаратов под итальянскими фамилиями (например, космический корабль Галилей в декабре этого года должен достичь Юпитера).

ПЛУТОН НАМ НЕ НУЖЕН

Члены подкомитета были единодушны в своих мнениях относительно планеты Плутон как не имеющей ни малейшего морального права на существование. "Перед нами планета, без которой мы решительно можем обойтись, — продолжал Дайлек О.Пойдеш, — Несколько лет назад она находилась в наибольшем удалении от Солнца — сейчас уже нет. Как прикажете понимать? Теперь нам говорят, что на самом деле это не одна планета, а две. Что, черт возьми, там происходит?!"

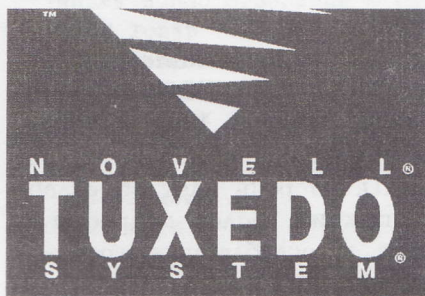
Резолюция теперь должна быть представлена на рассмотрение всего Конгресса, где, как полагают, она должна легко пройти — лишь меньшинство его членов имеют избирателей на затрону-

тых планетах.

Златкин из администрации НАСА поклялся, что будет противостоять дальнейшему урезанию Солнечной системы. "НАСА затратило значительные усилия на то, чтобы сделать планеты дешевле, быстрее и лучше. В результате даунсайзинга Солнечной системы большая часть этой работы окажется затраченной впустую," — сказал Златкин.

По мнению критиков, уменьшение числа планет не приведет к существенной экономии средств налогоплательщиков. Они обращают внимание на необходимость пересмотра учебников, в которых придется отразить новое положение вещей. К тому же, для реального устранения планет понадобятся определенные технические устройства.

Резолюция, вероятно, вызовет сильную оппозицию и со стороны религиозных фундаменталистов, которые всегда были настроены против уничтожения какой-либо из библейских планет.



Как ни странно, то, что надето на этих дамах, называется TUXEDO. Правда, экзотического фасона. В комплект поставки входят: недоуздок (halter), ремень (thong), гамаша (leggings) и обшлага (cuffs). Прелестные бантики на шее и



в некоторых других местах придают дамам вид миллионерш.

В общем, TUXEDO — это вещь! Продукция компании Steamed Heat International.

МАТЕРИАЛ НОМЕРА



Глеб Ладыженский

Системы управления базами данных — коротко о главном

Содержание

- 4. Обработка транзакций
 - 4.1. Понятие транзакции
 - 4.2. Мониторы транзакций
 - 4.2.1. Корпоративная среда обработки транзакций
 - 4.2.2. Модель обработки транзакций
 - 4.2.3. Интерфейс прикладного программирования АТМІ
 - 4.2.4. Функциональный подход
 - 4.2.5. Другие возможности TRM
 - 5. Средства защиты данных в СУБД
- Заклучение
Литература

Раздел 4. Обработка транзакций

4.1. Понятие транзакции

В Разделе 1 уже обсуждалось понятие транзакции. Транзакция представляет собой последовательность операторов языка SQL, которая рассматривается как некоторое неделимое действие над базой данных, осмысленное с точки зрения пользователя. В то же время, это логическая единица работы системы. Транзакция реализует некоторую прикладную функцию, например, перевод денег с одного счета на другой в банковской системе.

Существуют различные модели транзакций, которые могут быть классифицированы на основании различных свойств, включающих структуру транзакции, параллельность внутри транзакции, продолжительность и т.д. Чаще всего имеют в виду традиционные транзакции, характеризующиеся четырьмя классическими свойствами: атомарности, согласованности, изолированности, долговечности (прочности) — ACID (Atomicity, Consistency, Isolation, Durability). Иногда

традиционные транзакции называют ACID-транзакциями. Упомянутые выше свойства означают следующее.

1. Свойство **атомарности** выражается в том, что транзакция должна быть выполнена в целом или не выполнена вовсе.
2. Свойство **согласованности** гарантирует, что по мере выполнения транзакций данные переходят из одного согласованного состояния в другое — транзакция не разрушает взаимной согласованности данных.
3. Свойство **изолированности** означает, что конкурирующие за доступ к базе данных транзакции физически обрабатываются последовательно, изолированно друг от друга, но для пользователей это выглядит так, как будто они выполняются параллельно.
4. Свойство **долговечности** трактуется следующим образом: если транзакция завершена успешно, то те изменения в данных, которые были ею произведены, не могут быть потеряны ни при каких обстоятельствах (даже в случае последующих ошибок).

Расширенные транзакции допускают формирование из ACID-транзакций иерархических структур. Если конкретная модель ослабляет некоторые из требований ACID, то речь идет об ослабленной транзакции. Более подробно о расширенных и ослабленных транзакциях можно прочесть в [1].

В качестве примера, иллюстрирующего понятие транзакции, рассмотрим традиционную базу данных, состоящую из пяти таблиц: Заказчик, Менеджер, Отделение, Товар, Заказ, и содержащую информацию о поставках некоторого Товара (контролируемых Менеджером) по определенному Заказу из Отделения

компании Заказчику. Структура таблиц выглядит следующим образом:

- Заказчик (Номер_заказчика, Компания, Лимит_кредита)
- Менеджер (Номер_менеджера, Имя, Возраст, Должность, Место_работы, Время_работы, Руководитель, Квота, Объем_продаж)
- Отделение (Номер_отделения, Город, Число_служащих, Объем_продаж, Продаж_в_год)
- Товар (Серия, Индекс, Описание, Цена, Количество_на_складе)
- Заказ (Номер_заказа, Дата, Номер_заказчика, Индекс_продукта, Количество, Стоимость)

Рассмотрим транзакцию "Изменить количество в заказе номер 113051 с 4 до 10, причем стоимость заказа изменится с 140580 до 355000. Заказ на товар серии 'МП456' индекс '674A', контролируемый менеджером под номером 108, работающим в г.Тверь (отделение 21)". Предполагается, что пользователь вводит операторы SQL последовательно один за другим, пользуясь некоторым интерактивным средством ввода SQL-запросов.

```
UPDATE Заказ
SET Количество = 10, Стоимость =
355000
WHERE Номер_заказа = 113051
UPDATE Менеджер
SET Объем_продаж = Объем_продаж -
140580 + 355000
WHERE Номер_менеджера = 108
UPDATE Отделение
SET Объем_продаж = Объем_продаж -
140580 + 355000
WHERE Номер_отделения = 21
UPDATE Товар
SET Количество_на_складе =
Количество_на_складе + 4 - 10
WHERE Серия = 'МП456' AND Индекс =
'674A'
```

...К этому моменту никаких ошибок не обнаружено...

,COMMIT WORK

Последний оператор фиксирует изменения в базе данных, достигнутые в результате выполнения операторов SQL, составляющих транзакцию.

Вновь рассмотрим ту же самую транзакцию, но предположим, что в процессе ввода операторов SQL, ее составляющих, пользователь допустил ошибку и указал в последнем операторе ошибочную серию. Немедленно после этого не-

обходимо использовать оператор ROLLBACK WORK — и тогда все изменения в базе данных, декларированные операторами внутри транзакции, будут отменены и база данных вернется к состоянию на момент начала транзакции.

```
UPDATE Заказ
SET Количество = 10, Стоимость =
355000
WHERE Номер_заказа = 113051
UPDATE Менеджер
SET Объем_продаж = Объем_продаж -
140580 + 355000
WHERE Номер_менеджера = 108
UPDATE Отделение
SET Объем_продаж = Объем_продаж -
140580 + 355000
WHERE Номер_отделения = 21
UPDATE Товар
SET Количество_на_складе =
Количество_на_складе + 4 - 10
WHERE Серия = 'МП465' AND Индекс =
'674A'
```

...Ошибка! Вместо 'МП456' использовано 'МП465', требуется откатить изменения...

ROLLBACK WORK

Таким образом, возможны два варианта завершения транзакции. Если все операторы выполнены успешно, и в процессе выполнения транзакции не произошло никаких сбоев программного или аппаратного обеспечения, транзакция фиксируется.

Фиксация транзакции — это действие, обеспечивающее запись на диск изменений в базе данных, которые были сделаны в процессе выполнения транзакции. До тех пор, пока транзакция не зафиксирована, возможно аннулирование этих изменений, восстановление базы данных в то состояние, в котором она была на момент начала транзакции. Фиксация означает, что все результаты выполнения транзакции становятся постоянными. Они станут видимыми другим транзакциям только после того, как текущая транзакция будет зафиксирована. До этого момента все данные, затрагиваемые транзакцией, будут "видны" пользователю в состоянии на начало текущей транзакции.

Если в процессе выполнения транзакции случилось нечто такое, что делает невозможным ее нормальное завершение, база данных должна быть возвращена в исходное состояние. Откат транзакции — это действие, обеспечивающее

аннулирование всех изменений данных, которые были сделаны операторами SQL в теле текущей незавершенной транзакции.

Каждый оператор в транзакции выполняет свою часть работы, но для успешного завершения всей работы в целом требуется безусловное завершение их всех. Группирование операторов в транзакции сообщает СУБД, что вся эта группа должна быть выполнена как единое целое, причем такое выполнение должно поддерживаться автоматически.

В стандарте ANSI/ISO SQL определены модель транзакций и функции операторов COMMIT и ROLLBACK. Стандарт определяет, что транзакция начинается с первого SQL-оператора, инициируемого пользователем или содержащегося в программе. Все последующие SQL-операторы составляют тело транзакции. Транзакция завершается одним из четырех возможных способов (рис. 1):

- оператор COMMIT означает успешное завершение транзакции; его использование делает постоянными изменения, внесенные в базу данных в рамках текущей транзакции;
- оператор ROLLBACK прерывает транзакцию, отменяя изменения, сделанные в базе данных в рамках этой транзакции; новая транзакция начинается непосредственно после

использования ROLLBACK;

- успешное завершение программы, в которой была инициирована текущая транзакция, означает успешное завершение транзакции (как будто был использован оператор COMMIT);
- ошибочное завершение программы прерывает транзакцию (как будто был использован оператор ROLLBACK).

Существуют и другие модели транзакций (например, принятая в Sybase). Диалект SQL Sybase включает четыре оператора обработки транзакций:

- оператор BEGIN TRANSACTION сигнализирует о начале транзакции;
- оператор COMMIT TRANSACTION означает успешное выполнение транзакции; его смысл соответствует оператору COMMIT в модели ANSI/ISO, однако после его использования никакая новая транзакция не стартует;
- оператор SAVE TRANSACTION устанавливает точку сохранения в теле транзакции — то есть сохраняет состояние базы данных, достигнутое на момент использования оператора. Точки сохранения именованы, что позволяет устанавливать несколько таких точек в транзакции; имя точки сохранения

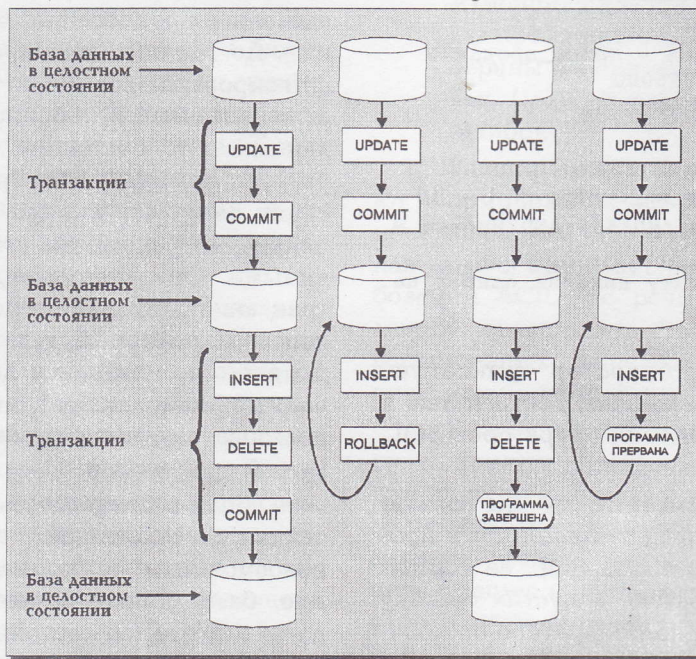


Рис.1. Модель транзакции ANSI/ISO.

указывается в операторе SAVE TRANSACTION.

- оператор ROLLBACK TRANSACTION выполняет две функции. Если в операторе указано имя точки сохранения, то производится откат к состоянию базы данных, достигнутому к моменту выполнения оператора SAVE TRANSACTION. Если же в операторе не указано имя точки сохранения, то производится откат изменений к моменту начала транзакции (см. рис. 2).

Точки сохранения применяются, как правило, в протяженных транзакциях и позволяют разделить транзакцию на несколько небольших осмысленных фрагментов. Пользователь может зафиксировать работу в любой точке транзакции с тем, чтобы выполнить ее откат к состоянию, соответствующему этой точке.

Откат и фиксация транзакций становятся возможными благодаря журналу транзакций. Он используется следующим образом.

Известно, что все операции над реляционной базой данных суть операции над строками таблиц. Следовательно, для обеспечения отката таблиц к предыдущим состояниям достаточно хранить не состояния всей таблицы, а лишь те ее строки, которые подверглись изменениям.

При выполнении любого оператора SQL, который модифицирует базу данных, СУБД автоматически заносит очередную запись в журнал транзакций. Запись состоит из двух компонентов: первый — это состояние строки до внесения изменений, второй — ее же состояние после внесения изменений. Только после занесения записи в журнал транзакций, СУБД действительно модифицирует базу данных. Если после данного оператора SQL был выполнен оператор COMMIT, то в журнале транзакций делается отметка о завершении текущей транзакции. Если же после оператора SQL следовал оператор ROLLBACK, то СУБД просматривает журнал транзакций и отыскивает записи, отражающие состояние измененных строк до модификации. Используя их, СУБД восстанавливает те строки в таблицах базы данных, которые

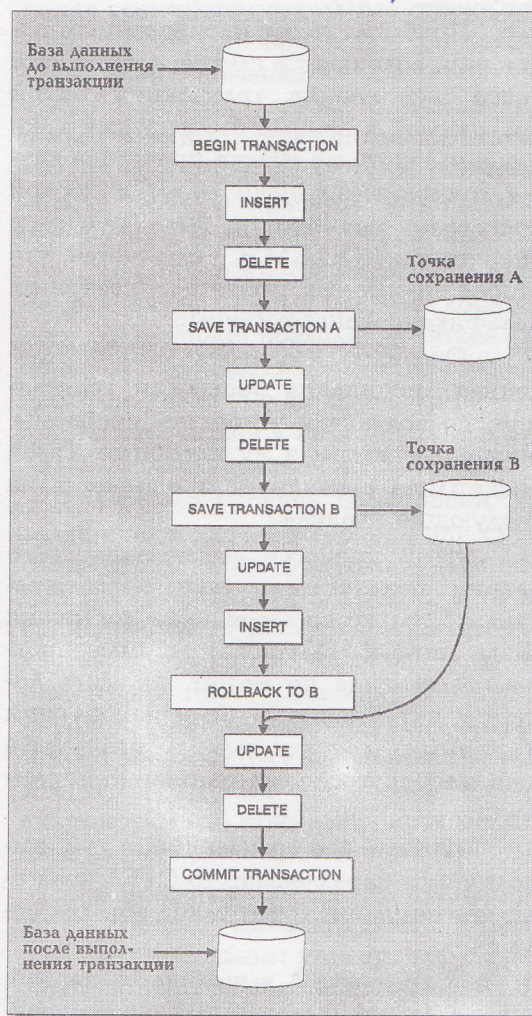


Рис.2. Модель транзакции с точками сохранения.

были модифицированы текущей транзакцией — таким образом аннулируются все изменения в базе данных.

Важные проблемы многопользовательских СУБД связаны с организацией с помощью механизма транзакций одновременного доступа множества пользователей к одним и тем же данным. Они (проблемы) кратко могут быть сформулированы как потеря изменений, незафиксированные изменения и ряд других, более сложных проблем.

Потеря изменений происходит в ситуации, когда две или несколько программ читают одни и те же данные, вносят в них какие-либо изменения и затем пытаются одновременно записать результат по прежнему месту. Разумеется, в базе данных могут быть сохранены изменения, выполненные только одной программой — другие изменения будут потеряны.

Проблема незафиксированных изменений возникает в случае, когда в процессе выполнения транзакции одной программой в данные были внесены изменения, которые тут-же прочитала другая программа, однако затем в первой программе транзакция была прервана оператором ROLLBACK. Получается, что вторая программа прочитала неверные, незафиксированные данные.

Очевидно, что необходима определенная дисциплина обработки транзакций, позволяющая устранить проблемы, описанные выше, и им подобные. Такая дисциплина существует и опирается на следующие правила:

(1) В процессе выполнения транзакции пользователь (или программа) "видит" только согласованные состояния базы данных. Пользователь (или программа) никогда не может получить доступ к незафиксированным изменениям в данных, достигнутым в результате действий другого пользователя (программы).

(2) Если две транзакции, А и В, выполняются параллельно, то СУБД полагает, что результат будет такой же, как если бы:

- транзакция А выполнялась первой, а за ней была выполнена транзакция В;
- транзакция В выполнялась первой,

а за ней была выполнена транзакция А.

Эта дисциплина известна как сериализация транзакций. Фактически она гарантирует, что каждый пользователь (программа), обращающаяся к базе данных, работает с ней так, как будто не существует других пользователей (программ), одновременно с ним обращающихся к тем же данным. Для практической реализации этой дисциплины большинство коммерческих СУБД используют механизм блокировок.

Для пояснения действия механизма блокировок воспользуемся рис.3. На нем представлены три таблицы базы данных (Заказ, Отделение, Товар), к которым возможен доступ в процессе выполнения двух транзакций — А и В. Когда при выполнении транзакции А происходит обращение к таблицам Заказ и Отделение, СУБД блокирует фрагмент таблицы (что имеется в виду под фрагментом, будет разъяснено ниже) до тех пор, пока транзакция не будет зафиксирована или отменена. Транзакция В выполняется параллельно и блокирует на момент своего выполнения таблицу Товар. Если в процессе выполнения транзакции В делается попытка получить доступ к заблокированным таблицам, обработка транзакции приостанавливается и возобновляется только после того, как

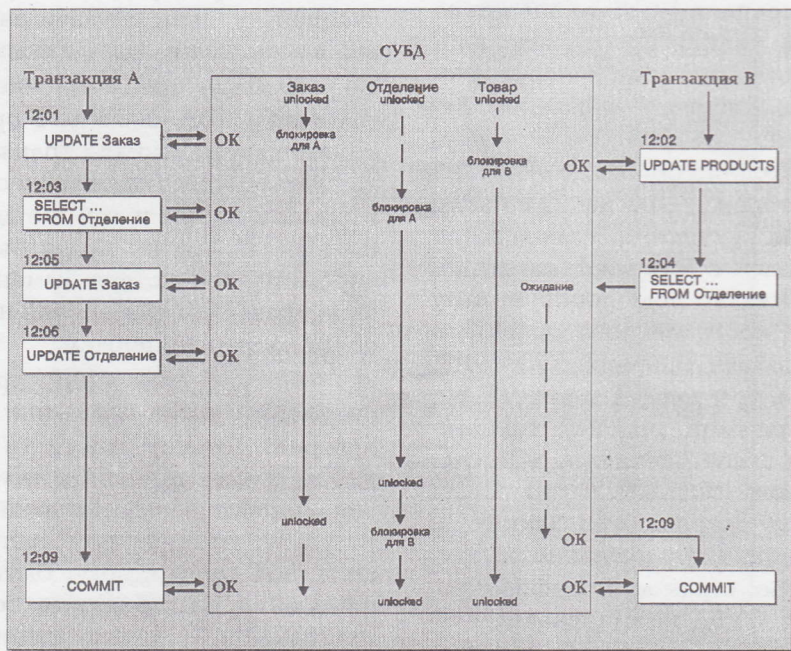


Рис.3. Конкурирующие транзакции и блокировки.

транзакция А завершается и освобождает блокированные ею таблицы.

Как мы видим, механизм блокировок разрешает проблемы, связанные с доступом нескольких пользователей (программ) к одним и тем же данным. Однако его применение связано с существенным замедлением обработки транзакций, вызванным необходимостью ожидания, когда освободятся данные, захваченные конкурирующей транзакцией. Можно попытаться минимизировать вызванные этим задержки, локализуя фрагменты данных, захватываемые транзакцией. Так, СУБД может блокировать всю базу данных целиком (очевидно, что это неприемлемый вариант), таблицу базы данных, часть таблицы, отдельную строку. Описанное выше называется уровнями блокировки. Современные СУБД используют, как правило, блокировки на уровне частей таблиц (страниц) и/или на уровне записей.

При блокировке на уровне страниц СУБД захватывает для выполнения транзакции некоторый фрагмент таблицы, запрещая доступ к нему конкурирующим транзакциям. Последние, впрочем, могут захватить другие страницы той же таблицы. Так как размер страниц обычно невелик (2-4 Кб), то время ожидания транзакций, конкурирующих за доступ к страницам таблицы, оказывается приемлемым даже для режима оперативного доступа к базе данных. Блокировка на уровне страниц реализована, например, в Ingres.

Если СУБД реализована таким образом, что может захватывать для выпол-

нения транзакции отдельные строки таблицы, то скорость обработки транзакции существенно повышается. Блокировка на уровне записей (строк) позволяет добиться максимальной производительности за счет того, что захватываемый объект (запись) является минимальной структурной единицей базы данных.

Теоретически, блокировка на уровне элементов данных (захват конкретного поля строки) позволит добиться еще большей производительности. Однако, насколько известно автору, пока ни в одной коммерческой СУБД не удалось реализовать этот уровень блокировки.

Помимо уровней блокировки, выделяют также тип блокировки или схему блокировки. Конкурирующие транзакции могут захватывать данные, в то же время разрешая доступ к этим данным другим транзакциям, но только для чтения. Кроме того, транзакции могут блокировать данные, не допуская захвата тех же данных другими транзакциями, в том числе и только для чтения. Проиллюстрируем это примером, представленным на рис. 4.

Две транзакции (А, В) конкурируют за доступ к таблицам Заказ, Отделение, Товар. Сравним этот пример с предшествующим. За счет использования более гибкой схемы блокировки для таблицы Отделение (разделяемая блокировка) мы получаем более высокий уровень параллелизма доступа транзакций А, В к одним и тем же данным.

Оператор SQL LOCK TABLE позволяет транзакции установить на таблицу определенный тип блокировки. Рассмот-

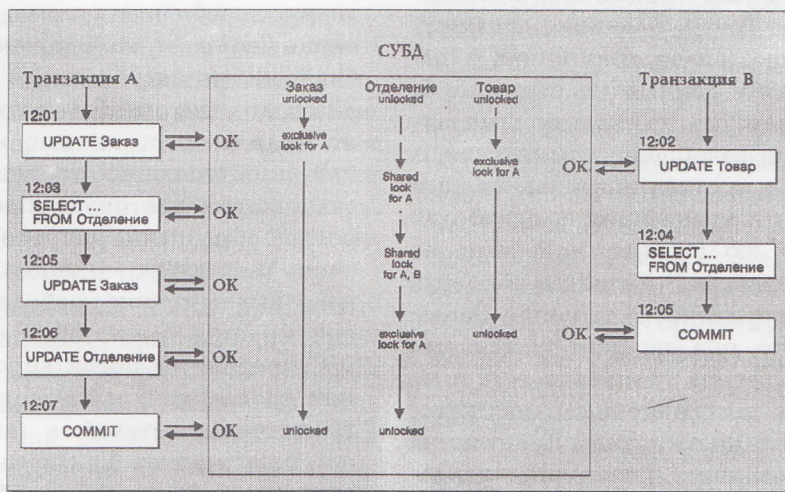


Рис.4. Схемы блокировок.

рим этот оператор на примере СУБД Informix.

Синтаксис оператора LOCK TABLE таков:

```
LOCK TABLE <имя таблицы> IN SHARE MODE
LOCK TABLE <имя таблицы> IN EXCLUSIVE
MODE
```

Если используется тип блокировки SHARE, то это означает, что в рамках данной транзакции (то есть транзакции, внутри которой был использован оператор LOCK TABLE) разрешено чтение из данной таблицы в других транзакциях (однако запись в таблицу запрещена). Использование EXCLUSIVE запрещает другим транзакциям читать и/или записывать данные в таблицу. Блокировка таблицы осуществляется внутри транзакции единожды. Это означает, что невозможно изменить тип блокировки внутри транзакции, например, с SHARE на EXCLUSIVE; это можно сделать, только по завершении транзакции:

```
BEGIN WORK
LOCK TABLE Заказ IN EXCLUSIVE MODE
...
COMMIT WORK
BEGIN WORK
LOCK TABLE Заказ IN SHARE MODE
...
COMMIT WORK
```

Внимательный читатель может обратить внимание на то, что транзакции могут попасть в тупиковую ситуацию, состояние неразрешимой взаимоблокировки. Оно иллюстрируется простейшей ситуацией. Транзакция А обновляет таблицу Заказ, блокируя некоторую ее страницу (для определенности — страницу С). В то же время транзакция В обновляет таблицу Товар, блокируя страницу D этой таблицы. Далее, транзакция А пытается обновить данные на странице D таблицы Товар (но, поскольку транзакция В еще не завершена, транзакция А переводится в состояния ожидания того момента, когда транзакция В освободит захваченную ею страницу). В этот же момент транзакция В пытается обновить данные на странице С таблицы Заказ. Сделать этого она не может, так как страница захвачена транзакцией А и не освобождена, поскольку последняя находится в состоянии ожидания. Транзакция В также переводится в состояния ожида-

ния. Налицо ситуация взаимоблокировки транзакций, которая может продолжаться бесконечно, если СУБД не предпримет специальные меры. На практике могут происходить более сложные взаимоблокировки нескольких транзакций.

Для их предотвращения СУБД периодически проверяет блокировки, установленные активными транзакциями. Если СУБД обнаруживает взаимоблокировки, она выбирает одну из транзакций, вызвавшую ситуацию взаимоблокировки, и прерывает ее. Это освобождает данные для внесения изменений конкурирующей транзакцией, разрешая тупиковую ситуацию. Программа, которая инициировала прерванную транзакцию, получает сообщение об ошибке, информирующее ее о причине прерывания (имела место тупиковая ситуация). Избежать их может и правильная стратегия внесения изменений в базу данных. Одним из наиболее простых и эффективных правил может быть следующее: все программы, которые обновляют одни и те же таблицы, должны, по мере возможности, делать это в одинаковой последовательности (UPDATE Заказ, UPDATE Отделение, UPDATE Товар...).

В современной литературе часто встречается термин OLTP (On-Line Transaction Processing), который обычно переводят как "оперативная обработка транзакций", то есть выполнение транзакций в режиме реального времени. Система OLTP обязана учитывать жесткие временные требования, следующие из специфики прикладной области. Например, процедура покупки и оформления авиабилета должна происходить очень быстро и не задерживать очередь. Система, регистрирующая продажи билетов, должна обрабатывать одновременно несколько сотен запросов (транзакций), поступающих от множества продавцов авиабилетов. Требование по скорости обработки запроса могут быть очень жесткими — менее секунды, однако вызваны они требованиями реальной жизни. Область OLTP обычно — это массивный поток коротких и простых транзакций, исходящих от сотен и тысяч пользователей, а также (возможно), базы данных большого объема. Если говорить о прикладных областях OLTP,

то это, прежде всего, центры кредитных карточек, системы резервирования авиабилетов и мест в отелях, телекоммуникационные системы и т.д.

Если данные хранятся в одной базе данных, то транзакция к ней рассматривается как локальная. Распределенные системы обычно включают несколько компьютеров — серверов баз данных, называемых узлами. Данные физически распределены между ними. На каждом узле располагается некоторая локальная база данных, содержащая фрагмент данных из общей, распределенной базы (см. предыдущий раздел). В распределенных базах транзакция, выполнение которой заключается в обновлении данных на нескольких узлах сети, называется глобальной или распределенной транзакцией.

Для пользователя распределенной базы данных глобальная транзакция выглядит как обычная. Это означает, что хотя в процессе выполнения распределенной транзакции происходит изменение данных в нескольких базах на разных узлах, сам этот процесс организован таким образом, что программисту, инициирующему обработку транзакции внутри своей программы, нет необходимости заботиться о синхронности завершения транзакции на каждом из узлов, ею затрагиваемых.

Внешне выполнение распределенной транзакции выглядит как обработка транзакции к локальной базе данных. Тем не менее, распределенная транзакция включает в себя несколько локальных транзакций, каждая из которых завершается двумя путями — фиксируется или прерывается. Распределенная транзакция фиксируется только в том случае, когда зафиксированы все локальные транзакции, ее составляющие. Если хотя бы одна из локальных транзакций была прервана, то должна быть прервана и распределенная транзакция. Как на практике учесть это требование?

В современных СУБД предусмотрен так называемый протокол двухфазовой (или двухфазной) фиксации транзакций (two-phase commit). Фаза 1 начинается, когда при обработке транзакции встретился оператор COMMIT. Сервер распределенной БД (или компонент

СУБД, отвечающий за обработку распределенных транзакций) направляет уведомление "подготовиться к фиксации" всем серверам локальных БД, выполняющим распределенную транзакцию. Если все серверы приготовились к фиксации (то есть откликнулись на уведомление и отклик был получен), сервер распределенной БД принимает решение о фиксации. Серверы локальных БД остаются в состоянии готовности и ожидают от него команды "зафиксировать". Если хотя бы один из серверов не откликнулся на уведомление в силу каких-либо причин, будь то аппаратная или программная ошибка, то сервер распределенной БД откатывает локальные транзакции на всех узлах, включая даже те, которые подготовились к фиксации и оповестили его об этом.

Фаза 2 — сервер распределенной БД направляет команду "зафиксировать" всем узлам, затронутым транзакцией, и гарантирует, что транзакции на них будут зафиксированы. Если связь с локальной базой данных потеряна в интервал времени между моментом, когда сервер распределенной БД принимает решение о фиксации транзакции и моментом, когда сервер локальной БД подчиняется его команде, то сервер распределенной БД продолжает попытки завершить транзакцию, пока связь не будет восстановлена.

4.2. Мониторы транзакций

Мониторы обработки транзакций (Transaction Processing Monitor — TPM), или, проще, мониторы транзакций — это программные системы (которые относят к категории middleware, то есть к посредническому или промежуточному программному обеспечению), решающие задачу эффективного управления информационно-вычислительными ресурсами в распределенной системе. Они представляют собой гибкую, открытую среду для разработки и управления мобильными приложениями, ориентированными на оперативную обработку распределенных транзакций. В числе важнейших характеристик TPM — масштабируемость, поддержка функциональной полноты и целостности приложений, до-

стижение максимальной производительности при обработке данных при невысоких стоимостных показателях, поддержка целостности данных в гетерогенной среде. ТРМ опираются на трехзвенную модель "клиент-сервер" (модель сервера приложений или AS-модель), описанную в Разделе 2. Естественно, что все преимущества модели отражаются и на программных системах, построенных на ее основе.

На современном рынке мониторов транзакций основными "действующими лицами" являются такие системы, как ACMS (DEC), CICS (IBM), TOP END (NCR), PATHWAY (Tandem), ENCINA (Transarc), TUXEDO Sytem (Novell). Наиболее известной из этой группы является система CICS — наверняка среди читателей найдутся специалисты, работавшие с ней на мэйнфрейме IBM. Несмотря на принципиальное сходство, конкретные ТРМ отличаются рядом характеристик, причем различия часто вытекают из специфики операционной системы, в которой реализован и функционирует ТРМ. Ниже будут описаны основные возможности ТРМ на базе операционной системы UNIX.

4.2.1. Корпоративная среда обработки транзакций

ТРМ на базе UNIX опирается на фундаментальное понятие — корпоративную среду обработки транзакций (Enterprise Transaction Processing — ETP).

Архитектура ETP (рис. 5) — это три ряда компьютеров:

- Ряд 1. Персональные станции (Personal Workstations);
- Ряд 2. Компьютеры под управлением ОС UNIX (UNIX Transaction Processing Servers — UPTS);
- Ряд 3. Mainframe-системы (Proprietary Transaction Processing Servers — PTPS) или компьютеры под управлением UNIX с RISC-архитектурой процессоров;

Разумеется, мы говорим о трехуровневой архитектуре систем, имея в виду лишь те организации, в которых уже длительное время используются мэйнфреймы. Они хранят огромные массивы данных, на них выполняются программные системы объемом в миллионы строк кода — и именно поэтому невозможно сразу отказаться от мэйнфреймов. Для современных организаций, которые избавлены от необходимости интегрировать в корпоративную информационную систему мэйнфреймы, поскольку у них их попросту нет, ситуация упрощается — и можно смело говорить не о трехуровневой, но о двухуровневой архитектуре ETP. Дело в том, что трехуровневая архитектура ETP характерна для Европы и США, где мэйнфреймы стали неотъемлемой частью информационных систем. Для России же мэйнфреймы — пройденный этап, мы начинаем создавать информационные системы "с чисто-

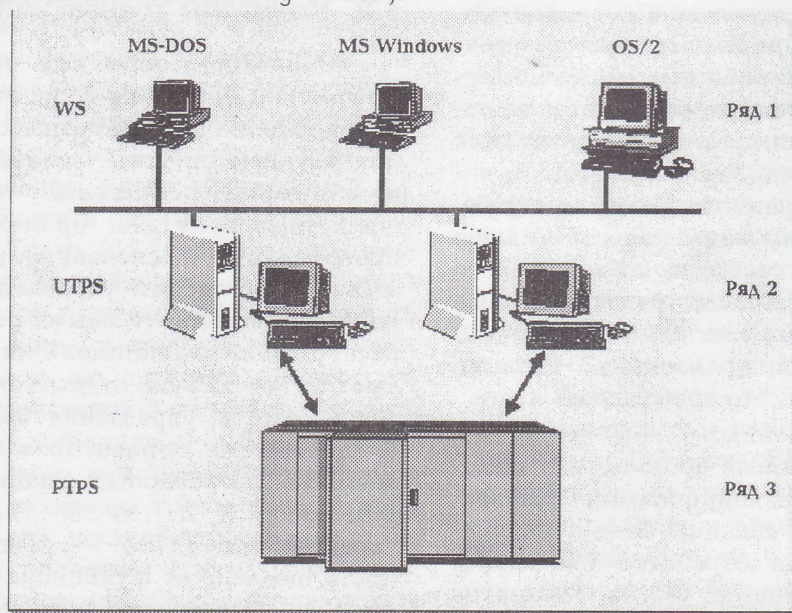


Рис.5. Промышленная среда обработки транзакций.

го места" ("персоналки" не в счет) и для отечественных организаций скорее будет характерна двухуровневая архитектура ЕТР. Так что схема, представленная на рисунке 5, отражает исторически сложившуюся архитектуру систем, и является вынужденным решением в тех ситуациях, когда организация имеет мэйнфреймы. Правильной является система из двух рядов компьютеров — ряда 1 и ряда 2. Не следует путать трехзвенную модель "клиент-сервер" (Раздел 2), касающуюся организации прикладной системы, с трехуровневой архитектурой ЕТР.

Компьютеры ряда 1, функционирующие под управлением DOS, MS Windows, OS/2, UNIX, используются в качестве рабочих мест конечных пользователей. Характерная черта ЕТР — отсутствие ограничений на модели компьютеров, составляющих этот ряд. Однако, как правило, ряд 1 состоит из компьютеров на базе процессоров Intel 486/Pentium под управлением MS Windows (MS Windows фактически стала стандартом оконного графического интерфейса для большинства категорий пользователей и стандартом операционной среды для подавляющего числа прикладных программ и систем).

Ряд 2 составляют компьютеры среднего класса под управлением ОС UNIX, на которых функционирует ядро ТРМ, и, как правило, реляционные СУБД (Oracle, Informix, Ingres), выступающие в качестве менеджера ресурсов. Кроме того, на них же может быть установлен шлюз к ТРМ в операционной среде mainframe (как правило, разработчики ТРМ на базе UNIX предусматривают в конфигурации своих систем шлюз к наиболее популярной такой системе — IBM CICS).

Ряд 3 представлен мэйнфреймами или RISC-компьютерами под управлением UNIX. О мэйнфреймах мы говорим в тех ситуациях, когда исторически сложилось так, что они существуют в организации уже долгое время, берут на себя большую часть всего объема обработки транзакций, концентрируют огромные вычислительные ресурсы и содержат большие массивы данных (то есть речь идет об унаследованных системах). Если этого "тяжелого наследия" нет, то можно

смело использовать в качестве компьютеров ряда 3 RISC-серверы, сегодня приближающиеся по производительности к мэйнфреймам.

Таким образом, среда обработки транзакций формируется из набора разнородных компьютеров (и соответствующих ОС), ранжируемых от персональных компьютеров до мэйнфрейм-систем. ТРМ на базе UNIX представляет собой своего рода "клей", который связывает вместе компьютеры трех рядов в открытую унифицированную среду обработки транзакций.

Ключом к интеграции систем, функционирующих на компьютерах различных рядов, является специализированный интерфейс прикладного программирования АТМІ (Application Transaction Manager Interface), обеспечивающий:

- для ряда 1 — формирование и передачу запросов от клиентов к серверам, выполняющимся на компьютерах ряда 2;
- для ряда 2 — обработку запросов, поступающих от компьютера ряда 1 (в том числе и с обращением к менеджеру ресурсов), и, при необходимости, формирование и направление запросов к серверам, выполняющимся на компьютерах ряда 3.
- для ряда 3 — обработку запросов, поступающих от серверов ряда 2.

Отметим, что подобное представление о корпоративной среде обработки транзакций не является абстракцией. Сегодня многие организации приходят именно к такой, "трехуровневой" архитектуре информационных систем (с той

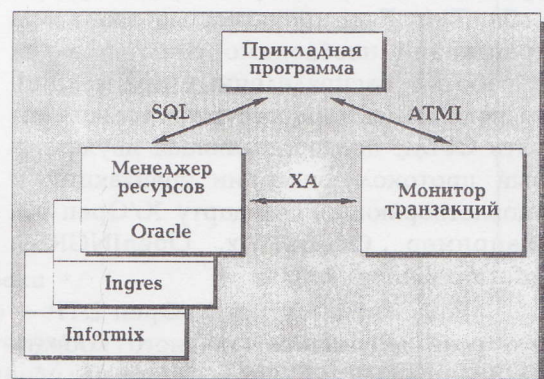


Рис.6. Модель обработки распределенных транзакций X/Open DTP.

оговоркой, что наличие ряда 3 вызвано историческими причинами — мэйнфреймы использовались первоначально и сразу от них отказаться невозможно).

4.2.2. Модель обработки транзакций

Понятия транзакции в ТРМ и в традиционных СУБД значительно отличаются. Суть остается одной, но в понимании СУБД транзакция — это атомарное действие над базой данных, в то время как в ТРМ транзакция трактуется гораздо шире. Она включает не только операции с данными, но и любые другие действия — передачу сообщений, выдачу отчетов, запись в индексированные файлы, опрос датчиков и т.д. Это позволяет реализовать в ТРМ прикладные транзакции, бизнес-транзакции, что в СУБД, вообще говоря, сделать невозможно.

ТРМ опирается на модель обработки распределенных транзакций X/Open DTP, которая описывает взаимодействие трех субъектов обработки транзакций — прикладной программы (в качестве прикладной программы фигурирует как сервер приложения, так и клиент приложения), менеджера транзакций (Transaction Manager — ТМ) и менеджера ресурсов (Resource Manager — RM). Модель представлена на рис. 6.

На RM возложено управление информационными ресурсами — будь то файлы, базы данных или что-то другое. Приложение взаимодействует с RM либо с помощью набора специальных функций, либо (если в качестве RM выступает реляционная SQL-ориентированная СУБД) посредством операторов языка SQL, инициируя необходимые операции с данными. Последние оформляются как транзакции, обработку которых берет на себя ТМ. Если с помощью монитора транзакций необходимо решать задачи обработки распределенных транзакций, то роль менеджера ресурсов должна играть СУБД, поддерживающая двухфазовый протокол фиксации транзакций и удовлетворяющая стандарту X/Open XA (например, Oracle 7.x, OpenINGRES, Informix-Online 7.x).

Роль ТМ в модели X/Open DTP — это роль диспетчера, главного координатора транзакций. Он обладает полным набором функций управления как локальными, так и глобальными, распре-

деленными транзакциями. В последнем случае транзакция может обновлять данные на нескольких узлах, причем управление данными на них, вообще говоря, осуществляется различными RM. Обработка распределенных транзакций обеспечивается за счет использования протокола двухфазовой фиксации транзакций, который гарантирует целостность данных в информационной системе, распределенной по нескольким узлам, независимо от того, какой RM управляет обработкой данных на каждом таком узле. Эта уникальная возможность как раз и позволяет рассматривать ТРМ как средство интеграции в гетерогенной информационной среде.

Функции ТМ в модели X/Open DTP не ограничиваются только управлением транзакциями. Он берет на себя также координацию взаимодействия клиента и сервера (поэтому иногда его называют менеджером транзакций и коммуникаций). При этом используется высокоуровневый интерфейс АТМІ, представляющий собой набор вызовов функций на языке третьего поколения (например, на языке С). С его помощью разработчик реализует один из нескольких режимов взаимодействия клиента и сервера в рамках расширенной модели "клиент-сервер". Ни сервер приложения, ни клиент приложения не содержат явных вызовов менеджера транзакций — они включены в библиотечные функции АТМІ и невидимы извне. Таким образом, детали взаимодействия прикладной программы и монитора транзакций скрыты от разработчика, что и дает основание говорить об АТМІ как о высокоуровневом интерфейсе.

Модель X/Open DTP не описывает в деталях структуру ТРМ. Она лишь определяет, из каких компонентов должна состоять любая система DTP и как эти компоненты взаимодействуют друг с другом. Будучи воплощенной в конкретной системе, модель дополняется возможностями, существенно расширяющими традиционные представления о технологии "клиент-сервер".

4.2.3. Интерфейс прикладного программирования АТМІ

Как уже говорилось выше, ТРМ — это инструмент эффективной реали-

зации технологии "клиент-сервер". Что конкретно за этим стоит? Реальная информационная система — это совокупность взаимодействующих программ. Некоторые из них предоставляют набор услуг, называемых обычно сервисами (перевод английского термина "services" — некоторое искажение правил

русского языка, где слово "сервис" обычно употребляется в единственном числе). Такие программы рассматриваются как программы-серверы. Программы, желающие воспользоваться предоставляемыми услугами и обращающиеся с этой целью к серверам, обычно называют клиентами. Интерфейс прикладного

```

/* Программа - клиент */
#include <stdio.h>
#include "atmi.h"

#define BUFLen 80

main()
{
    char *buf;
    long len;

    /* Инициализация связи клиента с сервером */
    if (tpinit((TPINIT *) NULL) == -1) { return(1);}
    /* Выделение памяти под буфер */
    if ((buf = tmalloc("STRING", NULL, BUFLen)) == NULL)
    {
        tpterm();
        return(2);
    }
    strcpy (buf, "TUXEDO example");
    /* Вызов сервиса CONVERT */
    if (tpcall("CONVERT", buf, 0, &buf, &len, 0) == -1)
    {
        fprintf (stderr, "Service request failed");
        tpfree (buf);
        tpterm();
        return(3);
    }
    /* Печать результата */
    printf ("Converted string %s", buf);
    /* Освобождение памяти, занятой под буфер */
    tpfree (buf);
    /* Отключение программы-клиента от сервера */
    tpterm ();
    return (0);
}

/* Программа - сервер */
#include <stdio.h>
#include "atmi.h"

CONVERT (TPSVCINFO *rqst)
{
    int i;

    /* Выполнить преобразование строки */
    for (i = 0; i < rqst->len; i++)
        rqst->data[i] = convert(rqst->data[i]);
    /* Вернуть преобразованный буфер */
    tpreturn (TPSUCCESS, 0, rqst->data, 0L, 0);
}

```

программирования описывает набор вызовов, с помощью которых можно организовать взаимодействие программы-клиента и программы-сервера несколькими различными способами. При этом надо иметь в виду, что программы эти выполняются в рамках различных процессов, которые, возможно, функционируют на различных, быть может, разнородных компьютерах. Следовательно, речь идет об описании межпроцессного взаимодействия в терминах "клиент-сервер"

В качестве иллюстрации приведем пример использования TUXEDO ATMI. Отметим, что она представляет интерес только для читателей, знакомых с языком С. Остальным мы рекомендуем пропустить этот раздел и перейти к чтению следующего.

В примере программа-клиент передает программе-серверу строку для конвертации. Для этого в программе-клиенте используется вызов `tpcall()`. С его помощью организуется синхронный режим взаимодействия клиента и сервера, когда клиент запрашивает сервер и моментально получает ответ. Задача сервера — выполнить некоторое преобразование передаваемой строки и вернуть результат клиенту.

Как мы видим в примере, обращение клиента к серверу происходит синхронно: клиент заполняет буфер данных, вызывает сервис "CONVERT"; данные в результирующем буфере возвращаются в программу-клиент немедленно по завершении работы сервиса "CONVERT" программы-сервера. Однако, это — простейший и самый очевидный способ организации взаимодействия клиента и сервера. Возможны и другие, более изощренные способы взаимодействия, например, полудуплексный режим обмена запросами между клиентом и сервером (в один момент времени только один процесс может посылать и только один процесс — получать сообщения).

4.2.4. Функциональный подход

Фундаментальная характеристика ТРМ — функциональный (function-centric) подход к проектированию бизнес-приложений. Еще раз отметим, что сосредоточение всех прикладных функций

в серверах приложений по сути означает "поставку (или предоставление) функций" (functions shipping) для программы-клиента, в отличие от традиционной архитектуры с сервером базы данных, следующей парадигме "поставка (или предоставление) данных" (data shipping).

Возможность декомпозиции приложений по нескольким уровням с четко очерченными функциями и стандартными интерфейсами позволяет создавать легко модифицируемые системы со стройной и целостной архитектурой. Концентрация чисто прикладных функций в серверах приложений и использование унифицированных интерфейсов с другими логическими компонентами делает прикладную систему практически полностью независимой как от конкретной реализации интерфейса с пользователем, так и от необходимого ей менеджера ресурсов. Первое означает, что для реализации интерфейса с пользователем может быть выбран практически любой удобный и привычный для разработчика инструментарий, будь то Microsoft Visual C++ или Visual Basic; следствием второго является то, что менеджер ресурсов (например, СУБД) может быть заменен на другой, поддерживающий тот же стандарт интерфейса с прикладной программой. Для реляционных СУБД в качестве унифицированного интерфейса используется встроенный (embedded) SQL. Разумеется, в реализации ESQL для каждой конкретной СУБД имеются различия, порой весьма существенные. Поэтому приложение должно быть либо разработано специально с целью работы с конкретной СУБД, либо оно должно быть спроектировано так, чтобы максимально безболезненно перенастраиваться на работу с другой СУБД.

Концентрация прикладных функций в сервере приложения на практике означает совершенно иной — по сравнению с другими видами программного обеспечения — подход к администрированию приложения, существенно упрощающий обновление бизнес-функций и контроль за их непротиворечивостью. Начнем с того, что разделение компонента представления и прикладного компонента позволяет поручить их реализацию двум практически независимым

специализированным группам разработчиков. Первая занимается разработкой и реализацией интерфейса с пользователем, уделяя основное внимание связанным с этим вопросам (стиль интерфейса, эргономические требования и т.д.). Вторая группа полностью концентрируется на "чистой" реализации приложения и не обращает никакого внимания на интерфейс с пользователем. Интерфейс между компонентами, разрабатываемыми разными группами, представляет собой специализированный API (например, TUXEDO ATMI). Преимущество подобного подхода — в специализации, разделении труда и более быстрой реализации проекта за счет параллельной работы групп программистов.

Функциональный подход имеет своим следствием и преимущества администрирования приложения. Отныне оно рассматривается как единое целое; сервер приложения имеет набор параметров, устанавливаемых его администратором; как и сервер БД, сервер приложения запускается и останавливается специальными командами; существуют команды, позволяющие опросить параметры сервера приложения и вывести их на консоль.

Любые изменения в прикладных функциях осуществляются локально, в сервере приложения, и никак не затрагивают коды клиентов приложения, которых (клиентов) может быть множество. Более того, эти изменения вносятся специалистом — администратором сервера приложения, единственным лицом, отвечающим за функциональность и семантическую целостность приложения. Разработчики программ-клиентов никак не участвуют в процедуре обновления приложения — это выходит за рамки их компетенции.

4.2.5. Другие возможности TRM

Уникальная возможность TRM — динамическая настройка параметров системы для достижения требуемой производительности (баланс загрузки). TRM поддерживает как статический, так и динамический баланс. Суть заключается в том, что TRM запускает или останавливает серверы приложений в зависимости от predetermined условий и текущего состояния системы. Для оптимизации

пропускной способности системы TRM тиражирует копии процессов-серверов на этом же или других узлах, предоставляя тем самым в распоряжение клиентов приложения необходимые вычислительные ресурсы (в виде дополнительных процессов) для выполнения запросов клиентов.

Помимо вертикального и горизонтального масштабирования, TRM обеспечивает так называемую матричную масштабируемость. Это — интеграция дополнительных ресурсов в гетерогенную среду в любой ее точке без изменения архитектуры приложения. Достигается это за счет динамической реконфигурации, что на практике означает, что в конфигурацию системы динамически, без остановки серверов приложений, может быть добавлен, например, новый сервер приложения, дополнительный менеджер ресурсов или новый компьютер. Очевидно, что матричная масштабируемость практически неограниченно расширяет возможности управления параметрами системы для достижения требуемой производительности.

TRM обладает возможностями, которые существенно снижают стоимость обработки данных в online-приложениях. Небольшие затраты на ее приобретение с лихвой компенсируются экономией на СУБД. Дело в том, что, как правило, стоимость современных СУБД рассчитывается исходя из числа одновременных подключений. Клиент считается подключенным к СУБД, начиная с момента открытия сеанса с базой данных и заканчивая его закрытием. В течение сеанса СУБД считает клиента активным и вынуждена хранить контекст его подключения, даже в том случае, если клиент вообще не направляет запросов СУБД, а выполняет свои внутренние функции, либо просто ждет ввода от пользователя.

Одна из основных функций TRM — обеспечение быстрой обработки запросов, поступающих к серверу приложений от множества клиентов (от сотен до тысяч). TRM выполняет ее, мультиплексируя запросы на обслуживание, направляя их серверам приложения, число которых контролируется им самим. Запросы на обработку данных формируются сервером приложения на языке SQL и адресуются СУБД. Между клиен-

том приложения и СУБД появляется дополнительный слой, который играет роль мультиплексора. Действительно, клиент подключается к серверу, передает ему данные для обработки; после того, как сервер выполнил требуемые действия, клиент получает результаты обработки и отключается. Контекст подключения не сохраняется, так как в этом нет никакой необходимости. В то же время клиент обращается с запросами на обслуживание не к СУБД, а к серверу, следовательно, СУБД регистрирует и отслеживает подключения сервера, но не клиента приложения. Однако таких подключений будет заведомо меньше, чем возможных подключений клиентов — хотя бы потому, что сервер приложений предоставляет сервис, разделяемый одновременно несколькими клиентами. Но это позволяет ограничить максимально возможное число одновременных подключений к СУБД, уменьшив тем самым ее стоимость.

Важнейшая характеристика ТРМ — поддержка многомашинных конфигураций с возможностью миграции серверов приложений и их групп на резервный компьютер в случае сбоев в работе основного — является фундаментом, на котором может быть построена система, по надежности близкая к абсолютной. Действительно, применение так называемых безотказных (fault tolerant) компьютеров гарантирует сохранение работоспособности лишь при случайных сбоях, но бессильно перед злоумышленником или в случае механического повреждения.

Раздел 5. Средства защиты данных в СУБД

Как только данные структурированы и сведены в базу данных, возникает проблема организации доступа к ним множества пользователей. Очевидно, что нельзя позволить всем без исключения пользователям беспрепятственный доступ ко всем элементам базы данных. В любой базе данных существует конфиденциальная информация, доступ к которой может быть разрешен лишь ограниченному кругу лиц. Так, в банковской системе особо конфиденциаль-

ной может считаться, например, информация о выданных кредитах.

Это — один из аспектов проблемы безопасности в СУБД, детальное обсуждение которой выходит за рамки статьи. Отметим, что в самом общем виде требования к безопасности реляционных СУБД формулируются так:

- Во-первых, данные в любой таблице должны быть доступны не всем пользователям, а лишь некоторым из них.
- Во-вторых, некоторым пользователям должно быть разрешено обновлять данные в таблицах, в то время как для других допускается лишь выбор данных из этих же таблиц.
- В-третьих, для некоторых таблиц необходимо обеспечить выборочный доступ к ее столбцам.
- В-четвертых, некоторым пользователям должен быть запрещен непосредственный (через запросы) доступ к таблицам, но разрешен доступ к этим же таблицам в диалоге с прикладной программой.

В этом разделе мы лишь кратко коснемся средств защиты данных в СУБД и методов авторизации доступа к данным, которые являются общепринятыми для большинства современных СУБД, а также обсудим новые методы защиты данных.

Схема доступа к данным во всех реляционных СУБД выглядит примерно одинаково и базируется на трех принципах:

- Пользователи СУБД рассматриваются как основные действующие лица, желающие получить доступ к данным. СУБД от имени конкретного пользователя выполняет операции над базой данных, то есть добавляет строки в таблицы (INSERT), удаляет строки (DELETE), обновляет данные в строках таблицы (UPDATE). Она делает это в зависимости от того, обладает ли конкретный пользователь правами на выполнение конкретных операций над конкретным объектом базы данных.
- Объекты доступа — это элементы базы данных, доступом к которым

можно управлять (разрешать доступ или защищать от доступа). Обычно объектами доступа являются таблицы, однако ими могут быть и другие объекты базы данных — формы, отчеты, прикладные программы и т.д. Конкретный пользователь обладает конкретными правами доступа к конкретному объекту.

- Привилегии (privileges) — это операции, которые разрешено выполнять пользователю над конкретными объектами. Например, пользователю может быть разрешено выполнение над таблицей операций SELECT (ВЫБРАТЬ) и INSERT (ВКЛЮЧИТЬ).

Таким образом, в СУБД авторизация доступа осуществляется с помощью привилегий. Установление и контроль привилегий — прерогатива администратора базы данных.

Привилегии устанавливаются и отменяются специальными операторами языка SQL — GRANT (РАЗРЕШИТЬ) и REVOKE (ОТМЕНИТЬ). Оператор GRANT указывает конкретного пользователя, который получает конкретные привилегии доступа к указанной таблице. Например, оператор

```
GRANT SELECT, INSERT ON Деталь TO
Битов
```

устанавливает привилегии пользователю Битов на выполнение операций выбора и включения над таблицей Деталь. Как видно из примера, оператор GRANT устанавливает соответствие между операциями, пользователем и объектом базы данных (таблицей в данном случае).

Привилегии легко установить, но легко и отменить (что особенно характерно для нашей страны). Отмена привилегий выполняется оператором REVOKE. Пусть, например, пользователь Битов утратил доверие администратора базы данных и последний решил лишить его привилегий на включение строк в таблицу Деталь. Он сделает это, выполнив оператор

```
REVOKE INSERT ON Деталь FROM Битов
```

Конкретный пользователь СУБД опознается по уникальному идентификатору (user-id). Любое действие над базой данных, любой оператор языка SQL

выполняется не анонимно, но от имени конкретного пользователя. Идентификатор пользователя определяет набор доступных объектов базы данных для конкретного физического лица или группы лиц. Однако он ничего не сообщает о механизме его связи с конкретным оператором SQL. Например, когда запускается интерактивный SQL, как СУБД узнает, от имени какого пользователя осуществляется доступ к данным? Для этого в большинстве СУБД используется сеанс работы с базой данных. Для запуска на компьютере-клиенте программы переднего плана (например, интерактивного SQL) пользователь должен сообщить СУБД свой идентификатор и пароль. Все операции над базой данных, которые будут выполнены после этого, СУБД свяжет с конкретным пользователем, который запустил программу.

Некоторые СУБД (Oracle, Sybase) используют собственную систему паролей, в других (Ingres, Informix) применяется идентификатор пользователя и его пароль из операционной системы.

Для современных баз данных с большим количеством пользователей актуальна проблема их объединения в группы. Традиционно применяются два способа определения групп пользователей.

Согласно первому, один и тот же идентификатор используется для доступа к базе данных целой группы физических лиц (например, сотрудников одного отдела). Это упрощает задачу администратора базы данных, так как достаточно один раз установить привилегии для этого "обобщенного" пользователя. Однако такой способ в основном предполагает разрешение на просмотр, быть может, на включение, но ни в коем случае — на удаление и обновление. Как только идентификатор (и пароль) становится известен большому числу людей, возникает опасность несанкционированного доступа к данным посторонних лиц.

Другой способ заключается в том, что конкретному физическому лицу присваивается уникальный идентификатор. В этом случае администратор базы данных должен позаботиться о том, чтобы каждый пользователь получил собственные привилегии. Если количество поль-

зателей базы данных возрастает, то администратору становится все труднее контролировать привилегии. В организации, насчитывающей свыше 100 пользователей, решение этой задачи потребует от него массу внимания. Легко ошибиться при назначении привилегий конкретному пользователю, а ведь бывает и так, что администратор базы данных, как и сапер, ошибается один раз.

Современные СУБД позволяют исправить эти неудобства, предлагая третий способ администрирования (Ingres, Informix). Суть его состоит в поддержке, помимо идентификатора пользователя, еще и идентификатора группы пользователей. Каждый пользователь, кроме собственного идентификатора, имеет также идентификатор группы, к которой он принадлежит. Чаще всего группа пользователей соответствует структурному подразделению организации, например, отделу. Привилегии устанавливаются не только для отдельных пользователей, но и для их групп.

Одна из проблем защиты данных возникает по той причине, что с базой данных работают как прикладные программы, так и пользователи, которые их запускают. В любой организации существует конфиденциальная информация о заработной плате ее служащих. К ней имеет доступ ограниченный круг лиц, например, финансовый контролер. В то же время к этой информации также имеют доступ некоторые прикладные программы, в частности, программа для получения платежной ведомости. Тогда на первый взгляд может показаться, что ее может запускать только финансовый контролер. Если он отсутствует, то сделать это может любой рядовой служащий — при условии, что ему известен пароль финансового контролера. Таким образом, необходимость запуска некоторых прикладных программ пользователями, которые обладают различными правами доступа к данным, приводит к нарушению схемы безопасности.

Одно из решений проблемы заключается в том, чтобы прикладной программе также были приданы некоторые привилегии доступа к объектам базы данных. В этом случае пользователь, не обладающий специальными привилегия-

ми доступа к некоторым объектам базы данных, может запустить прикладную программу, которая имеет такие привилегии.

Так, в СУБД Ingres это решение обеспечивается механизмом ролей (role). Роль представляет собой именованный объект, хранящийся в базе данных. Роль связывается с конкретной прикладной программой для придания последней привилегий доступа к базам данных, таблицам, представлениям и процедурам базы данных. Роль создается и удаляется администратором базы данных, ей может быть придан определенный пароль. Как только роль создана, ей можно предоставить привилегии доступа к объектам базы данных.

Пусть, например, в некоторой организации работает служащий, имеющий имя пользователя Битов. По характеру своей работы он часто обращается к таблице Заработная плата. Он также является членом группы Учет. Для пользователей этой группы разрешено выполнение операции SELECT над таблицей Заработная плата. Всякий раз, когда ему необходимо выполнить операцию выборки из таблицы, он должен для начала сеанса ввести идентификатор своей группы.

Однако, ни один из пользователей группы Учет не может непосредственно выполнить операцию UPDATE. Для этого необходимо запустить программу Контроль заработной платы, которая имеет привилегии обновления этой таблицы и выполняет специальные проверки для корректного выполнения операции. С этой целью администратором БД создается и помещается в БД роль Обновить заработную плату, для чего используется оператор

```
CREATE ROLE Обновить заработную
плату WITH PASSWORD = 'ДТ3110';
```

Оператор

```
GRANT SELECT, UPDATE ON Заработная
плата
```

```
TO ROLE Обновить заработную плату;
```

предоставляет новой роли привилегии на выполнение операций выбора и обновления таблицы Заработная плата. Когда пользователь Битов запускает программу Контроль заработной платы, то она получает привилегии роли Обновить зара-

ботную плату. Тем самым данный пользователь, сам не обладая привилегиями на обновление таблицы, тем не менее может выполнить эту операцию, но только запустив прикладную программу. Она же, в свою очередь, должна играть определенную роль, которой приданы соответствующие привилегии доступа к таблице.

Выше речь шла о реализациях схемы безопасности, которые ограничиваются моделью "данные-владелец". В них пользователь, работающий с базой данных, является владельцем некоторых ее объектов, а для доступа к другим объектам должен получить привилегии. Он может получить статус пользователя СУБД и некоторые привилегии доступа к объектам БД, войти в группу пользователей и получить соответствующие привилегии доступа, получить права на запуск некоторых прикладных программ. Это — так называемый добровольный или дискреционный контроль доступа (discretionary access control). Называется он так потому, что владелец данных по собственному усмотрению ограничивает круг пользователей, имеющих доступ к данным, которыми он владеет.

Несмотря на то, что в целом этот метод обеспечивает безопасность данных, современные информационные системы требуют другую, более изощренную схему безопасности — обязательный или принудительный контроль доступа (mandatory access control). Он основан на отказе от понятия владельца данных и опирается на так называемые метки безопасности (security labels), которые присваиваются данным при их создании. Каждая из меток соответствует некоторому уровню безопасности. Метки служат для классификации данных по уровням. Например, для правительственных и коммерческих организаций такая классификация выглядит следующим образом (см. рис. 7).

Так как данные расклассифицированы по уровням безопасности метками, конкретный пользователь получает ограниченный доступ к данным. Он может оперировать только с данными, расположенными на том уровне секретности, который соответствует его статусу. При этом он не является владельцем данных.

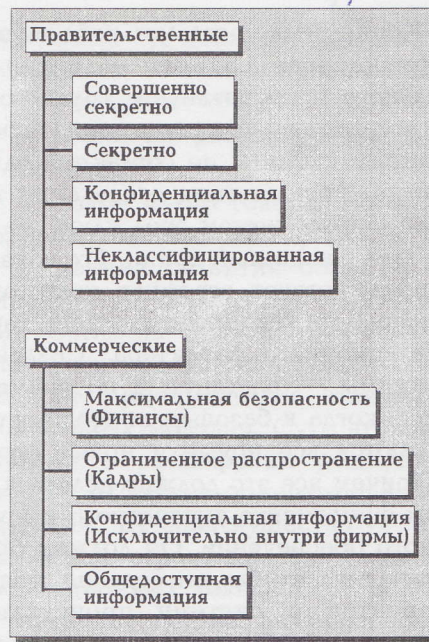


Рис.7. Классификация данных по уровням безопасности.

Эта схема безопасности опирается на механизм, позволяющий связать метки безопасности с каждой строкой любой таблицы в базе данных. Любой пользователь может потребовать в своем запросе отобразить любую таблицу из базы данных, однако увидит он только те строки, у которых метки безопасности не превышают уровень его компетенции.

Это означает, например, что строки таблицы, отмеченные как строки уровня максимальной безопасности, может увидеть только тот пользователь, у которого уровень безопасности — наивысший. Пользователи определенного уровня секретности могут видеть строки таблицы, отмеченные для их уровня безопасности, равно как и для всех уровней ниже данного. СУБД проверяет уровень безопасности пользователя и, в ответ на его запрос, возвращает только те строки таблицы, которые удовлетворяют запросу и соответствуют этому уровню.

По оценкам экспертов, концепция многоуровневой безопасности в ближайшие годы будет использована в большинстве коммерческих СУБД.

Заключение

Завершая обсуждение современных СУБД, приведем некоторые соображения по практическим аспектам их использования в качестве основы информационной системы.

Конкретная реально работающая информационная система — это плод длительного и кропотливого труда большого коллектива специалистов. Информационные системы не создаются моментально (предложения о создании информационных систем "под ключ" — не более чем рекламный трюк). Их появлению предшествует огромная подготовительная работа как аналитического, так и организационного характера. Создание же системы — растянутый по времени процесс, когда к базовым возможностям добавляются все новые и новые функции. Причем все это должно делаться на едином фундаменте аппаратного и программного обеспечения. Он должен быть заложен так, чтобы интеграция новых компонентов в систему происходила максимально безболезненно и не нарушала концепции и архитектуры всей системы.

К сожалению, в реальной жизни все складывается иначе. Обычно первоначально приобретается программное обеспечение, которое удовлетворяет сиюминутным требованиям. Как только возникает необходимость добавления новых функций и возможностей, выясняется, что они не поддерживаются. Разумеется, можно подумать о приобретении программ, способных решить вновь возникшие задачи. Но, увы — они не стыкуются с уже существующими системами. Раз так, приходится загружать работой программистов, которые пишут различные "нашлепки" и "заплатки". В результате вместо здания со стройной и целостной архитектурой получается ветхая конструкция со множеством подпорок и пристроек.

Слишком много отечественных организаций следует по этому пути, польстившись на кажущуюся дешевизну частных решений. Очень кстати здесь вспомнить народную мудрость, гласящую, что мы не столь богаты, чтобы делать дешевые покупки.

Чтобы избежать этих ошибок, стоит начать с возведения фундамента — базового программного обеспечения, и, в первую очередь, с СУБД. Каким требованиям современной быстро развивающейся организации она должна удовлетворять?

1. Организация будет расти, и к информационной системе будут подключаться все новые и новые сотрудники. Следовательно, система должна быть многопользовательской.
2. По мере расширения организация будет приобретать новые, не обязательно однотипные компьютеры. Значит, СУБД должна функционировать на множестве моделей компьютеров различных производителей, причем прикладные программы, разработанные для одной платформы, можно было бы без труда перенести на другую.
3. База данных организации будет непрерывно расти и расширяться. Следовательно, СУБД должна обеспечивать обработку и хранение больших объемов данных и поддерживать быстрорастущие БД.
4. В процессе развития информационной системы для реализации новых функций могут потребоваться различные механизмы обработки данных. Некоторые из них не обязательно потребуются сегодня, но непременно будут востребованы завтра и станут жизненно необходимыми послезавтра. Следовательно, СУБД должна быть многофункциональной. Именно поэтому столь много внимания уделено в книге механизмам активного сервера БД.
5. Для расширения информационной системы могут потребоваться новые компьютеры и новые программные системы. Следовательно, СУБД должна поддерживать как общепринятые стандарты сетевого обмена (TCP/IP, DECnet, IPX/SPX, NetBIOS, SNA и т.д.), так и стандарты межпрограммных интерфейсов (ATMI, XA, ODBC).
6. Возможно, что в организации появятся филиалы. Они будут работать с локальными БД. Значит, возникнет потребность объединения локальных БД в распределенную базу данных. Следовательно, СУБД должна управлять распределенными базами данных.

Разработчики СУБД учитывают эти требования. В настоящее время стан-

Перед Вами один из номеров бесплатного периодического информационно-технического бюллетеня "Jet Info", издаваемого компанией "Инфосистемы Джет" - ведущим российским системным интегратором в области UNIX-систем.

Если Вы еще не являетесь подписчиком "Jet Info", но хотели бы получать его регулярно, просим Вас отправить в наш адрес заполненный купон этого объявления. Отправьте этот купон и в том случае, если Вы сменили свой почтовый адрес или решили изменить форму доставки Вам бюллетеня.

Информация для подписчиков: не забудьте в течение каждого декабря и июля обязательно отправлять нам этот отрывной купон для перерегистрации.



1. Организация _____

2. Фамилия, имя, отчество и должность лица, в чей адрес будет осуществляться рассылка _____

3. Индекс и почтовый адрес _____

4. Телефон и/или факс _____

5. Адрес электронной почты _____

6. В какой форме Вы хотели бы получать бюллетень

печатный вариант по почте

электронную версию по электронной почте

7. Прежняя форма и адрес подписки (для подписчиков, меняющих адрес и форму подписки, либо тех, кто проходит перерегистрацию) _____

8. Общее число компьютеров в организации _____

9. Ваша организация использует

Вычислительную технику

Apple

DEC Alpha

DEC VAX

IBM PC

HP 9000

Silicon Graphics

Sun Microsystems

другое _____

Операционные системы

MS-DOS (Windows)

Novell NetWare

SCO Unix

Solaris / SunOS

UnixWare

VAX VMS

другое _____

10. Материалы на какую тему заинтересовали бы Вас в первую очередь _____

Мы будем рады, если кто-то из Вас захочет принять участие в подготовке выпусков "Jet Info" в качестве автора. Будем также признательны за любые конструктивные замечания и предложения по всем аспектам подготовки, выпуска и распространения нашего бюллетеня.

Редакция "Jet Info"