

ЛИНТЕР®



Технический обзор

Содержание

| | | |
|--------------|---|-----------|
| I. | ЗНАКОМСТВО С СИСТЕМОЙ ЛИНТЕР® | 4 |
| A. | ЧТО ТАКОЕ ЛИНТЕР® И ЧТО МОЖЕТ ЛИНТЕР® | 4 |
| B. | УТИЛИТЫ СИСТЕМЫ ЛИНТЕР® | 5 |
| II. | ЛИНТЕР® И ОПЕРАЦИОННОЕ ОКРУЖЕНИЕ | 7 |
| A. | МОБИЛЬНОСТЬ | 7 |
| B. | ПЕРЕМЕННЫЕ СРЕДЫ ОКРУЖЕНИЯ | 7 |
| C. | ФАЙЛЫ И ФАЙЛОВАЯ СИСТЕМА | 9 |
| D. | МНОГОЗАДАЧНАЯ СРЕДА | 9 |
| III. | КОНФИДЕНЦИАЛЬНОСТЬ ИНФОРМАЦИИ | 11 |
| A. | УПРАВЛЕНИЕ ДОСТУПОМ К ИНФОРМАЦИИ | 11 |
| 1. | <i>Авторизация пользователей</i> | 11 |
| 2. | <i>Контроль доступа к информации (ядро безопасности)</i> | 11 |
| 3. | <i>Дискреционная защита</i> | 11 |
| 4. | <i>Привилегии безопасности</i> | 11 |
| 5. | <i>Привилегии доступа</i> | 12 |
| 6. | <i>Использование представлений для управления доступом</i> | 12 |
| 7. | <i>Иерархия прав доступа</i> | 12 |
| 8. | <i>Мандатная защита</i> | 12 |
| 9. | <i>Контроль доступа с удаленных станций</i> | 13 |
| 10. | <i>Протоколирование работы</i> | 13 |
| 11. | <i>Контроль за хранением информации</i> | 13 |
| 12. | <i>Удаление остаточной информации</i> | 14 |
| B. | ПОДДЕРЖАНИЕ ВЫСОКОЙ ГОТОВНОСТИ ИНФОРМАЦИИ | 14 |
| IV. | СРЕДСТВА РЕАЛЬНОГО ВРЕМЕНИ СУБД ЛИНТЕР® | 15 |
| V. | ВНУТРЕННИЕ ЯЗЫКОВЫЕ СРЕДСТВА (SQL, ХРАНИМЫЕ ПРОЦЕДУРЫ, ТРИГГЕРЫ) | 16 |
| A. | ЯЗЫК ЗАПРОСОВ SQL | 16 |
| B. | ЯЗЫК ХРАНИМЫХ ПРОЦЕДУР СУБД ЛИНТЕР® | 17 |
| C. | ТРИГГЕРЫ: СВЯЗЬ СОБЫТИЙ И ХРАНИМЫХ ПРОЦЕДУР | 19 |
| D. | ОТЛАДКА ХРАНИМЫХ ПРОЦЕДУР ЛИНТЕР® | 19 |
| VI. | НАДЕЖНОСТЬ СИСТЕМЫ | 21 |
| A. | СИСТЕМНЫЙ ЖУРНАЛ | 21 |
| 1. | <i>Режимы транзакций СУБД ЛИНТЕР®</i> | 21 |
| 2. | <i>Иерархия транзакций в СУБД ЛИНТЕР®</i> | 23 |
| B. | АРХИВИРОВАНИЕ ТАБЛИЦ (БАЗЫ ДАННЫХ) | 24 |
| C. | ГОРЯЧЕЕ РЕЗЕРВИРОВАНИЕ | 27 |
| D. | TESTDB – УТИЛИТА ПРОВЕРКИ ФИЗИЧЕСКИХ СТРУКТУР БАЗЫ | 27 |
| VII. | ПРОИЗВОДИТЕЛЬНОСТЬ | 28 |
| VIII. | МАСШТАБИРУЕМОСТЬ | 29 |
| IX. | СОВМЕСТИМОСТЬ | 30 |
| X. | ЛИНТЕР® В СЕТЕВЫХ УСЛОВИЯХ | 32 |
| A. | РЕЖИМ «КЛИЕНТ-СЕРВЕР» | 32 |
| B. | РЕЖИМ «МУЛЬТИСЕРВЕР» | 32 |
| C. | РАСПРЕДЕЛЕННАЯ БАЗА ДАННЫХ | 33 |
| XI. | ОСНОВНЫЕ ХАРАКТЕРИСТИКИ СИСТЕМЫ | 34 |

| | | |
|--------------|---|-----------|
| XII. | АСИНХРОННАЯ РЕПЛИКАЦИЯ..... | 36 |
| A. | ПРАВИЛА РЕПЛИКАЦИИ | 36 |
| B. | СЕРВЕР РЕПЛИКАЦИИ СУБД ЛИНТЕР | 37 |
| C. | АНАЛИЗ ОСОБЕННОСТЕЙ СИСТЕМЫ. | 38 |
| D. | ДОСТОИНСТВА И НЕДОСТАТКИ | 38 |
| E. | НАПРАВЛЕНИЕ РАЗВИТИЯ | 39 |
| XIII. | СИСТЕМА СИНХРОНИЗАЦИИ DBSYNC | 40 |
| XIV. | БУДУЩЕЕ СИСТЕМЫ ЛИНТЕР® | 42 |
| A. | МОБИЛЬНОСТЬ | 42 |
| B. | МАСШТАБИРУЕМОСТЬ | 42 |
| C. | КОНФИДЕНЦИАЛЬНОСТЬ ИНФОРМАЦИИ | 42 |
| D. | ПРОИЗВОДИТЕЛЬНОСТЬ (ОПТИМИЗАЦИЯ, ИНДЕКСЫ) | 42 |
| E. | СЕТЕВЫЕ КОМПОНЕНТЫ..... | 42 |
| F. | АРХИВАТОР – LNB | 43 |
| G. | РЕАЛЬНОЕ ВРЕМЯ | 43 |
| H. | ПОЛНОТЕКСТОВАЯ ИНДЕКСАЦИЯ | 43 |
| XV. | ИНФОРМАЦИЯ О РАЗРАБОТЧИКЕ | 44 |
| A. | ТОВАРНЫЕ ЗНАКИ | 44 |
| B. | ИНТЕЛЛЕКТУАЛЬНАЯ СОБСТВЕННОСТЬ | 44 |
| C. | О ДОКУМЕНТЕ | 44 |
| D. | КОНТАКТНАЯ ИНФОРМАЦИЯ..... | 45 |
| E. | ТЕХНИЧЕСКАЯ ПОДДЕРЖКА | 45 |

I. Знакомство с системой ЛИНТЕР®

A. Что такое ЛИНТЕР® и что может ЛИНТЕР®

ЛИНТЕР – это система управления базами данных, обеспечивающая поддержку реляционной модели данных автоматизированных систем управления различного назначения, *систем реального времени* и систем, где необходимы повышенные требования к *надёжности, безопасности и секретности* данных.

В соответствии с реляционной моделью данные базы логически представлены в виде двумерных таблиц, что обеспечивает высокую степень независимости пользовательских программ от физического представления данных и удобство для неподготовленного пользователя.

Данные в таблицах физически хранятся построчно. В одну строку могут входить данные разных типов (символы, целые и вещественные числа, строки символов различной длины, и т.д.).

ЛИНТЕР позволяет выполнять следующие действия:

- удалять/изменять/добавлять объекты базы (данные, индексы, таблицы, хранимые процедуры, триггеры);
- вводить/изменять/удалять ограничения целостности данных;
- использовать полный набор возможностей стандартного языка SQL;
- работать с большими (до **2-ух гигабайт**) байтовыми объектами (**BLOB**);
- импортировать/экспортировать данные из/в ASCII и DBF файлов;
- блокировать/деблокировать доступ к таблице/записи;
- использовать (в приложениях и хранимых процедурах) различные режимы обработки транзакций;
- организовывать (и использовать) гибкую и надёжную систему безопасности и секретности информации (сертифицирован Государственной технической комиссией при Президенте РФ на соответствие 2 классу защиты информации от несанкционированного доступа, что соответствует уровню **В3** по американскому национальному стандарту orange book);
- сохранять/восстанавливать базу данных целиком или некоторые её объекты выборочно, устанавливать расписание и алгоритмы сохранения;
- транслировать запросы (с параметрами и без) и использовать уже оттранслированные запросы для ускорения работы приложения;
- создавать, отлаживать и запускать хранимые процедуры и триггеры;
- использовать возможности реального времени (приоритеты выполнения транзакций, асинхронное выполнение запросов, отслеживание процессов, проходящих в системе, приостановка и полная остановка работы указанной транзакции и пр.).

Последние два пункта, несомненно, важны при подготовке многозадачной прикладной системы и дают возможность пользователю настроить ЛИНТЕР на конкретное приложение и максимально ускорить работу системы.

- ЛИНТЕР использует унифицированные средства доступа к данным из программ пользователей. Их основой является Call-интерфейс. Он лежит в основе всех прочих программных интерфейсов.

В. Утилиты системы ЛИНТЕР®

Структура программного обеспечения ЛИНТЕР показана на следующем рисунке:

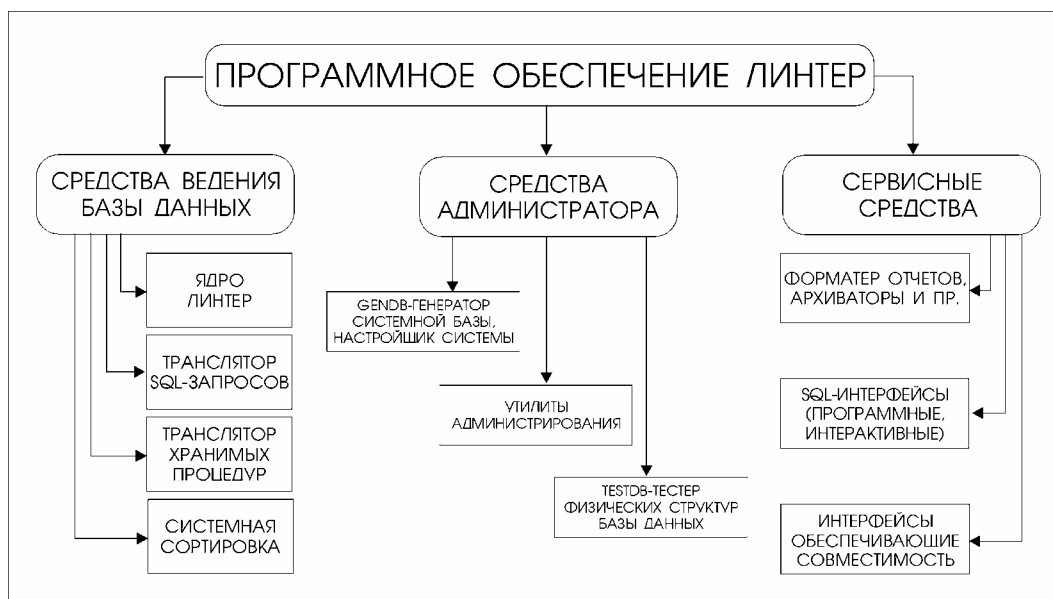


Рисунок 1 Состав программного обеспечения СУБД ЛИНТЕР®

В дистрибутив ЛИНТЕР включены следующие компоненты:

- **ядро СУБД ЛИНТЕР** (собственно ядро системы, транслятор с SQL, процессор сортировки, компилятор хранимых процедур, сетевые драйверы, менеджер распределённых транзакций);
- **программы обслуживания базы данных** (генератор системной базы данных, тестер физических структур);
- **организующие интерфейсы** (инструментарий администратора, менеджер хранимых процедур со встроенным отладчиком, интерактивный SQL-интерфейс);
- **средства разработки приложений** (встроенный SQL для C/C++, исполняющая система 4GL языка Intcom, средство интерактивной разработки Лакуна);
- **средства сохранения/восстановления данных** (в том числе «горячее» архивирование, быстрая загрузка/выгрузка всей базы данных или отдельных её частей и т.п.);
- **средства миграции данных** (импорт из DBF, ODBC-средство миграции и т.п.);
- **интерфейсы различного уровня** (ODBC-драйвер, интерфейс прямого доступа к ЛИНТЕР из Delphi/Kylix/C++ Builder, интерфейс для Java программ, API-интерфейс ЛИНТЕР, Call-интерфейс и т.п.);

ЛИНТЕР имеет множество сервисных средств, включающее разнообразные интерфейсы и системы программирования прикладных программ:

- Интерактивный/пакетный SQL-интерфейс – **Inl**.
- Инструментарии администратора (экранный – **Ldba**, командный – **Adm**, графический – **Lindesk**), позволяющий получить любую доступную информацию о состоянии базы/СУБД и произвести любые доступные действия.

- Средство разработки приложений – **ЛАКУНА**, для описания объектов приложения (документов, отчетов, меню и др.) и программных средств манипуляции этими объектами (процедуры обработки, определение событий объекта и способы их обработки, средства генерации отчетов, управление правами доступа к объектам и др.).
- Алгоритмический язык разработки приложений – **Intcom**, предоставляющий средства создания прикладных систем пользователя и обеспечивающий широкое использование ЛИНТЕР в непромышленной сфере (системы делового применения, информационно-поисковые системы, и т.п.).
- **PCI** – встроенный SQL (embedded SQL) для языков C и C++.
- **ODBC** – интерфейс ODBC 3.x.
- **OLE DB** – интерфейс доступа к данным в среде Windows.
- **Perl** – интерфейс совместимости с языком Perl.
- **Php** – интерфейс, позволяющий осуществлять доступ ко всем ресурсам СУБД ЛИНТЕР из программ написанных на PHP.
- **dbExpress** – интерфейс для прямого доступа к СУБД ЛИНТЕР из популярных средств разработки Delphi/Kylix/C++ Builder.
- **Jdbc** – интерфейс для JDBC 1.0, 2.0, 3.0.
- **Lintcl** – интерфейс для поддержки tcl/tk.
- **LinPy** – интерфейс для доступа к данным из Python.
- **Oralin** – интерфейс для использования СУБД ЛИНТЕР из программ, разработанных с использованием OCI интерфейса СУБД Oracle.
- **Встроенный SQL**, для использования запросов непосредственно в программах на языках C и Pascal. Подобное использование SQL гораздо удобнее и нагляднее для программиста, кроме того, использование встроенного SQL освобождает программиста от лишней рутинной работы.
- **LinAPY**-интерфейс (LINter Application Program Interface) – это интерфейс еще более низкого уровня, предназначенный для подготовки сложных программ на языке C. В программах, использующих вызовы этого интерфейса, можно использовать оттранслированные, асинхронные запросы, приоритеты запросов и т.п.

Этот обзор содержит описание базовой версии системы ЛИНТЕР, т.е. того, что не зависит от конкретной программно-аппаратной платформы.

Особенности работы системы на конкретной ЭВМ в конкретной операционной среде можно найти в:

- файле **Readme**,
- **Help-файлах** утилит системы,
- в документации.

II. ЛИНТЕР® и операционное окружение

A. Мобильность

ЛИНТЕР работает на многих программно-аппаратных платформах:

MS WINDOWS NT/2000/XP/2003 Server, MS WINDOWS 95/98, Windows CE, Embedded Linux (Zaurus), PalmOS, Linux, MCBC, FreeBSD, ИИТРОС, UNIXWare, UNIX SYSTEM V, SINIX, SUN Solaris, Digital UNIX, USIX, OS/9000, OS-9, QNX 4, QNX 6, VAX/VMS, OpenVMS, HP-UX, OC2000, Novell NetWare, MS DOS, AIX, IRIX, VXWorks.

Кроме того, уместно отослать читателя к гл. XIV «*Будущее системы ЛИНТЕР*».

На всех платформах базовый вариант системы ЛИНТЕР работает *внешне одинаково*. Это первое требование мобильности – идентичность внешних интерфейсов системы на всех платформах.

Таким образом, пользователь, переходя с одной платформы на другую, оказывается в привычной для себя обстановке.

Было бы неправильно думать, что ЛИНТЕР не использует преимущества, которые дает внешним интерфейсам та или иная платформа. ЛИНТЕР их использует. Однако в базовом варианте системы применяется только тот минимум, который присутствует на всех платформах. Этим можно объяснить полную идентичность интерфейсов систем, работающих на разных машинах, в разных операционных средах.

Кроме внешнего интерфейса, ЛИНТЕР мобилен также и по программному интерфейсу. При переносе прикладной информационной системы на другую платформу не возникнет проблем, связанных с программными интерфейсами (Call-интерфейсом, **LinAPI** и **Pci**).

Следующий уровень мобильности – мобильность по данным. ЛИНТЕР лишь частично удовлетворяет этому уровню, т.к. использует особенности арифметического процессора конкретной платформы (*целые и вещественные числа с плавающей точкой аппаратно зависимы*), а также особенности кодировки символов.

Другие типы (числа с фиксированной точкой, дата) машинно-независимы, и с ними проблем мобильности у пользователя не возникнет.

Совместимость по данным – наиболее важный момент при работе в разнородной сети. При работе в сетях СУБД ЛИНТЕР полностью обеспечивает совместимость данных с той техникой и операционной системой, которая находится на узле сети, запросившем эти данные.

B. Переменные среды окружения

При загрузке системы ядро ЛИНТЕР должно определить, где находится база, с которой хочет работать пользователь. Как дать понять системе, с какой из баз данных ядро будет работать в данный момент?

Это делается через *переменные среды окружения*. Такие переменные могут обозначать (содержать) имя физического устройства, путь в иерархическом дереве файловой структуры до нужного каталога, и т. п.

Переменные среды окружения – полезный элемент операционной системы, который рекомендуется использовать, чтобы делать программы менее зависимыми от физического расположения файлов.

Ядро СУБД ЛИНТЕР при загрузке сначала определяет наличие переменной среды окружения с именем

SY00

Эта переменная должна содержать путь до главных файлов базы данных, её системных таблиц.

Если специально не был указан (при помощи ключа **/BASE** при запуске системы) другой каталог, то ядро системы свяжется именно с той базой, главные файлы которой находятся в каталоге, обозначенном **SY00**.

При отсутствии в указанном **SY00** каталоге соответствующих файлов, ядро сообщит об ошибочной ситуации и прекратит работу.

Если переменная с именем **SY00** не определена, ядро будет считать, что база данных находится в *специально указанном* (при запуске системы) или в текущем *каталоге*.

Переменные среды во многих операционных системах могут быть локальные (известные только процессам, запущенным с конкретного терминала) и глобальные (известные всем). Программе, обращающейся за данными к СУБД, не обязательно знать, где расположены данные, что означает **SY00**, и т.д. Эта переменная среды окружения необходима ядру ЛИНТЕР, тестеру структур базы **Testdb**, а также **Gendb** – генератору системной базы.

При создании таблицы можно указать место, где нужно расположить файлы таблицы. Указывая конкретное место на диске (дисках) для файлов таблицы, пользователь должен понимать, что переменные среды окружения, содержащие обозначение этого места, должны быть известны ядру системы.

Эти переменные должны быть известны системе не только на этапе создания таблицы, но и при следующих запусках системы, т.к. их имена попали в каталоги базы, и при обращении к таблице поиск ее файлов будет выполняться по содержимому этих переменных.

Если пользователь не указывает место расположения файлов таблицы, ядро разместит их самостоятельно либо на устройстве **SY00** (при наличии), либо в каталоге по умолчанию (для ядра).

Обратите внимание, СУБД ЛИНТЕР имеет возможности использования различных кодировок и данных в UNICODE. Поэтому в систему введена переменная

LINTER_CP

для указания кодировки (по умолчанию) клиентского приложения.

Кроме **SY00**, утилиты системы используют еще переменную

LINTER_BIN

указывающую путь до местоположения дистрибутивных утилит ЛИНТЕР. Кроме них в этом каталоге находятся справочные файлы. Если в системе не установлена переменная с именем **LINTER_BIN**, то информация для справок будет искаться в текущем каталоге.

Интерактивный интерфейс – **Inl** использует для редактирования текущего запроса один из редакторов операционной среды. Спецификация командной строки запуска нужного редактора должна содержаться в переменной

LINTER_EDIT

C. Файлы и файловая система

СУБД ЛИНТЕР хранит каждую таблицу в виде набора файлов и пользуется всеми возможностями, которые предоставляет файловая система операционной системы.

Так, в многопользовательских/сетевых системах доступность файлов базы данных для пользователя (минуя доступ через СУБД) отслеживается операционной системой. Поэтому для ограничения доступа к файлам базы данных нужно, чтобы запуск ядра осуществлял только один пользователь.

Еще одна возможность, которой пользуется ЛИНТЕР, – это расширяемость файлов. Эта возможность позволяет тратить меньше усилий на расчет размера файлов таблицы, реже модифицировать ее физическую структуру. Но чрезмерное увлечение этой особенностью может привести к малоэффективному использованию дискового пространства и, конечно, к уменьшению скорости.

Если пользователь имеет точное представление об объеме и характере данных прикладной системы, то лучше однажды выделить место под файл. Последний при этом окажется «более непрерывным» и, следовательно, более эффективным при работе с СУБД.

Файлы операционной системы, используются не только ядром, но и утилитами ЛИНТЕР.

Ядро ЛИНТЕР неформально использует файловый процессор операционной системы, добавляя к нему еще и свой аппарат, который более осведомлен о характере процессов ввода/вывода из файлов базы.

Ядро ведет свой пул элементов ввода/вывода и использует свои алгоритмы буферизации/блокировок наряду с тем, что делает операционная система. Сочетание алгоритмов, учитывающих особенности СУБД, и универсальных алгоритмов операционной системы дает хорошие результаты.

D. Многозадачная среда

ЛИНТЕР – открытая система, предназначенная для использования именно в многозадачных операционных средах. Поэтому алгоритмам распараллеливания обработки запросов уделялось пристальное внимание.

При этом максимально используется то распараллеливание, которое дает операционная система, и в дополнение к этому ядро СУБД проводит собственное распараллеливание обработки запросов.

Алгоритм обработки запросов ведется несколькими задачами (ядро, транслятор запросов, транслятор хранимых процедур, сортировщик (и) ответов), распараллеливанием работы которых занимается операционная система.

Ядро, как реентерабельная программа примет пришедший на обработку запрос, поставит его в ряд параллельно обрабатываемых запросов и, *квантуя* обработку этих запросов, будет переключаться с одного запроса на другой.

Широкие возможности по настройке ЛИНТЕР включают и настройки, позволяющие сделать конкретную многозадачную прикладную систему более эффективной.

Более того, в системе предусмотрены средства слежения за распараллеливанием, которое проводит ядро. Пользуясь ими можно точнее установить эффективные параметры настройки системы или понять, что и в какой момент мешает общему потоку запросов, поступающих от пользовательских задач.

Средства слежения позволят системному аналитику написать программу сбора статистики (профиль) работы СУБД, получить полезные рекомендации и сделать конкретные выводы для улучшения настройки ЛИНТЕР.

Обратите внимание, что параллельно могут обрабатываться не только запросы, посланные из разных задач, но также и запросы одной задачи.

III. Конфиденциальность информации

Как известно политика безопасности – это совокупность норм, правил и практических приемов, которые регулируют управление, защиту и распределение ценной информации.

Направление развития функциональности защиты информации см. гл. XIV «*Будущее системы* ЛИНТЕР».

В СУБД ЛИНТЕР политика безопасности реализуется с помощью двух основных подсистем:

- подсистема управления доступом к информации,
- подсистема поддержания высокой готовности информации.

A. Управление доступом к информации

В СУБД ЛИНТЕР реализовано большое количество способов управления доступом к информации:

1. Авторизация пользователей

производится при установлении соединения с системой. Проверке подлежит регистрационное имя пользователя и его секретный пароль. Если процесс авторизации пользователя прошел успешно, то все дальнейшие запросы к СУБД по данному соединению однозначно связываются с данным пользователем.

2. Контроль доступа к информации (ядро безопасности)

проходит любой запрос на доступ к любым объектам базы данных. При необходимости, отметка о прохождении запроса (удачном/неудачном) протоколируется в *журнале системы защиты*. При этом используются критерии дискреционной и мандатной защиты, а так же проверяется возможность работы данного пользователя с конкретной клиентской станции.

3. Дискреционная защита

в СУБД ЛИНТЕР реализована с помощью аппарата привилегий, которые можно подразделить на две категории: *привилегии безопасности* (позволяют выполнять административные действия) и *привилегии доступа* (определяют права доступа конкретных субъектов к определенным объектам).

4. Привилегии безопасности

Таких привилегий (категорий пользователей) три:

- *Администратор базы данных – категория DBA*. Это управляющий созданием БД, ее конфигурированием, регистрацией пользователей, групп, ролей, записью регистрационной информации и т.п.
- *Привилегированные пользователи БД – категория RESOURCE*. Это пользователи, которые имеют право на создание собственных объектов БД и управление привилегиями доступа к ним.
- *Пользователи БД – категория CONNECT* оперируют с объектами БД в рамках выделенных им привилегий доступа.

5. Привилегии доступа

- **SELECT** – на выборку данных,
- **INSERT** – на добавление данных,
- **DELETE** – на удаление данных,
- **UPDATE** – на обновление данных,
- **ALTER** – на изменение параметров таблицы,
- **INDEX** – на создание/удаление индексов,
- **ALL** – включает все вышеперечисленные права доступа.

Эти привилегии может присваивать/изымать только владелец соответствующих объектов:

- таблиц,
- представлений,
- синонимов.

Привилегии можно объединять в *роли*.

6. Использование представлений для управления доступом

позволяет сделать видимыми для пользователя только определенные данные таблиц. Не предоставляя субъектам прав доступа к базовым таблицам, и сконструировав подходящие представления, администратор базы защитит таблицы от несанкционированного доступа и снабдит каждого пользователя своим видением базы данных.

7. Иерархия прав доступа

позволяет реализовать следующие виды ограничения доступа:

- *операционные ограничения* (за счет прав доступа SELECT, INSERT, UPDATE, DELETE, применимых ко всем или только некоторым столбцам таблицы);
- *ограничения по значениям* (за счет механизма представлений);

8. Мандатная защита

предназначена для построения информационных систем с высокой степенью защищённости и состоит в назначении различных уровней ценности для всей хранимой информации.

Для этого в СУБД ЛИНТЕР используются *метки доступа*. Метка доступа состоит из трех частей: *группы доступа* (именованная совокупность пользователей) и *двух уровней доступа*.

Для субъектов базы данных они называются: *уровень доверия пользователя* (WAL) и *уровень доступа пользователя* (RAL). Для объектов базы данных они называются: *уровень чтения данных* (RAL) и *уровень доступа данных* (WAL).

Метки доступа могут быть назначены всем субъектам базы и объектам: начиная от таблиц и *до полей записей* включительно.

9. Контроль доступа с удаленных станций

или сопоставление пользователя с устройством позволит учитывать различный уровень защищенности самих клиентских станций. При правильной политике безопасности ЛИНТЕР не допустит использование канала связи, в котором ценная информация может быть скомпрометирована.

Основопологающим понятием в процессе сопоставления пользователя с устройством является понятие *сетевого устройства*. Сетевым устройством в ЛИНТЕР считается любое устройство, имеющее уникальный идентификатор – адрес в сети. Каждое сетевое устройство в ЛИНТЕР характеризуется целой совокупностью параметров (адрес устройства, тип сети, тип подсети, маска разрешенного времени доступа, уровни мандатного доступа, маска разрешенных групп).

Разрешенные времена доступа могут быть указаны вплоть до временных интервалов в течение дня.

10. Протоколирование работы

или система слежения ЛИНТЕР применяется для контроля функционирования подсистемы защиты, обнаружения попыток несанкционированного доступа, исправления их последствий и предотвращения их в будущем.

В СУБД ЛИНТЕР производится протоколирование широкого спектра событий. Для того чтобы информация об определенном событии заносилась в журнал, необходимо выставить флаг протоколирования этого события.

В журнал системы безопасности заносятся следующая информация:

- отметка времени;
- имя пользователя;
- имя объекта;
- группа события;
- тип события;
- статус завершения ЛИНТЕР.

Сюда же заносится дополнительная информация о клиентской станции (сетевой адрес, PID клиента, сокет клиента), с которой пришел запрос.

Регулярный мониторинг журнала системы безопасности позволяет поддерживать надежность системы защиты на высоком уровне и своевременно реагировать на попытки обойти систему защиты.

11. Контроль за хранением информации

со стороны СУБД ЛИНТЕР позволит учитывать различный уровень защищенности внешних устройств постоянного хранения информации для размещения таблиц данных и временных рабочих файлов.

Например, часть таблиц может быть расположена на диске сервера, находящегося в охраняемом помещении, а часть размещена на жестком диске другой ЭВМ или даже на гибком диске. Защищенность устройства в последнем случае гораздо более низкая, и, следовательно, здесь не может быть расположена секретная информация.

Доступ к устройству в ЛИНТЕР может быть разрешен/запрещен различным группам пользователей. Кроме того, устройству назначается метка доступа, характеризующая его уровень защищенности и ограничивающая степень секретности содержащейся на нем информации.

В СУБД ЛИНТЕР каждое устройство характеризуется следующей информацией:

- идентификатором устройства;
- именем устройства;
- физическим путем к устройству;
- описанием устройства;
- параметрами доступа к устройству.

12. Удаление остаточной информации

закрывает еще один канал нелегального доступа к охраняемым данным – анализ остаточной информации. Анализ поддается как информация в оперативной памяти, так и информация во внешней памяти.

Для предотвращения этого в обоих случаях освободившееся пространство очищается с помощью записи маскирующей информации.

В. Поддержание высокой готовности информации

Помимо механизмов управления доступом к информации в СУБД ЛИНТЕР существуют механизмы поддержки высокой готовности информации.

Под этим подразумевается обеспечение доступа к информации в режиме реального времени, гарантированная независимость целостности и сохранности информации от сбоев в системе, от попыток разрушения информации, от аварийных ситуаций различного рода и т.д.

IV. Средства реального времени СУБД ЛИНТЕР®

В ЛИНТЕР есть несколько особенностей, выделяющих эту СУБД из ряда аналогов. Это свойства, по которым ЛИНТЕР можно отнести к системам реального времени.

Во-первых, возможность подачи запросов в асинхронном режиме. Отметим, что это именно полнокровная асинхронность с определением **процедуры обработки ответа**, которая включится только тогда, когда программа будет прервана пришедшим от ЛИНТЕР ответом на запрос.

Во-вторых, ЛИНТЕР может **обрабатывать запросы в соответствии с установленными для них приоритетами**. Более важные (приоритетные) будут выполнены раньше низкоприоритетных, им будут отданы системой все возможные ресурсы и т.п.

В-третьих, **аппарат событий** ЛИНТЕР позволит приложению устанавливать **особые ситуации** и обеспечивать реакцию на их возникновение. Например, какая-то задача прикладной системы SQL-запросом устанавливает событие А (например, модификация данных). Другие задачи могут запросить, чтобы их оповестили о возникновении события А. По возникновению этого события, запросившие его задачи будут прерваны, включатся соответствующие процедуры обработки ответа (на запрос об оповещении). По окончании обработки события (например, после того, как перезапрошены изменённые данные) программа продолжится с того места, где она была прервана.

В-четвёртых, возможность отделения этапа трансляции запроса от этапа его выполнения, т.е. запрос можно один раз оттранслировать, а затем многократно выполнять, наполняя его каждый раз новым константным содержанием (BIND параметров).

Это особенно удобно в программах сбора информации. При этом можно сочетать выполнение оттранслированного запроса и асинхронный режим его выполнения, что очень важно в системах управления технологическими процессами (например, при сборе информации с датчиков и занесения их базу данных).

Далее отметим возможность слежения из приложения за состоянием использования ресурсов ядра СУБД. Более того, имеется возможность получать статистику по физическому/логическому вводу-выводу и, следовательно, обладать информацией об интенсивности работы системы в тех или иных условиях работы приложения.

Это позволяет написать задачу с супервизорскими функциями. Такая задача производит слежение за процессами, происходящими в ядре ЛИНТЕР, и может решить, что обработка какого-то запроса требует слишком много ресурсов, и приостановить или прервать его обработку.

V. Внутренние языковые средства (SQL, хранимые процедуры, триггеры)

A. Язык запросов SQL

Язык SQL-ЛИНТЕР реализует международный стандарт языка SQL – ANSI/ISO SQL-92.

В SQL-ЛИНТЕР пользователь найдет такие мощные языковые средства, как предложение UNION, полный набор операций соединения – JOIN, все описанные в указанном стандарте возможности по реализации ограничений целостности и пр.

Для совместимости с некоторыми СУБД других производителей (Oracle, DB2, Informix, Microsoft SQL Server и др.) в язык запросов ЛИНТЕР введены специальные встроенные функции, языковая работа по управлению контролем доступа к информации, иерархические запросы к таблицам, последовательности и т.д.

Для удобства пользователей в SQL-ЛИНТЕР включены так же следующие расширения указанного стандарта:

1. Языковая работа с BLOB-столбцами.
2. Языковая работа с событиями в ЛИНТЕР.
3. Разрешено использование нескольких таблиц во FROM в операциях UPDATE и DELETE. Например,

DELETE FROM таблица **JOIN** список_таблиц **WHERE ...**

UPDATE таблица **JOIN** список_таблиц **WHERE ...**

4. Разрешена конструкция INTO в SELECT-операторе для совместимости с некоторыми диалектами языка SQL. Например,

SELECT список_выражений **INTO** список_параметров **FROM ...**

5. Разрешена конструкция **CAST NULL AS тип**.
6. Введены следующие предложения для установки режимов работы каналов:

SET TRANSACTION READ ONLY – перевод канала в режим read-only;

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED – перевод канала в режим грязного чтения.

7. Введены предложения для работы с правилами репликации:

CREATE REPLICATION RULE правило **FOR** таблица **TO** таблица **ON**
NODE имя_узла **USER**
 пользователь **PASSWORD** 'пароль' [**ENABLE** | **DISABLE**];

ALTER REPLICATION RULE правило
 [**PASSWORD** 'пароль'] [**ENABLE** | **DISABLE**];

DROP REPLICATION RULE правило;

8. Разнообразные возможности ALTER TABLE по модификации структуры таблицы – от изменения имен (таблицы, её столбцов) до изменений важнейших характеристик самой таблицы и её столбцов (например, размеров, числа файлов,

места их расположения, а для столбцов – длины данных, значений по умолчанию и т.д.).

Ещё один немаловажный штрих – возможность поиска отдельных слов и словосочетаний в больших текстовых значениях, что очень важно при организации больших полнотекстовых информационных баз.

В. Язык хранимых процедур СУБД ЛИНТЕР®

Наличие механизма *хранимых процедур* в СУБД ЛИНТЕР позволяет существенно расширить возможности языка SQL, организуя процедурную обработку данных на сервере согласно алгоритму пользователя.

По функциональной мощности хранимые процедуры ЛИНТЕР в некоторых аспектах даже превышают стандарт ANSI/ISO SQL-92/PSM (Persistent Stored Modules).

Есть множество важных моментов, не отраженных в стандарте, например:

- использование оттранслированных запросов и запросов с параметрами (динамически изменяемых запросов),
- управление транзакциями.

Хранимые процедуры ЛИНТЕР могут использовать как обычные, так и оттранслированные запросы с параметрами и без параметров, кроме того процедуры имеют возможность возвращать курсор, что очень удобно для программирования.

Язык хранимых процедур предоставляет мощный синтаксис выражений включающий все необходимые операции с переменными и значениями каждого типа данных, вызовы разнообразных стандартных функций (таких, как преобразование типов, работа со строчными данными и т.д.) операцию присваивания (тот факт, что присваивание является операцией, а не отдельным оператором, позволяет строить, например, такие конструкции: **a := b := c := 0;**).

Язык хранимых процедур позволяет работать со всеми стандартными типами данных ЛИНТЕР (**Integer, Smallint, Char, Byte, Numeric, Real, Double, Date**), а также дополнительным типом **BOOL** (логический). Тип **CHAR** рассматривается как строки с заданной максимальной длиной, и для него определены удобные в использовании строковые константы и операции конкатенации.

Все операции со всеми типами данных реализуют трехзначную логику, то есть поддерживается значение **NULL** для любого типа данных, которое означает состояние «значение не определено».

Последовательность операторов позволяет закодировать линейный алгоритм. Для организации ветвящихся алгоритмов может использоваться оператор типа **IF..ELSEIF...ELSEIF...ELSE..ENDIF**, оператор выбора **CASE**, оператор перехода на метку **GOTO**. Циклические алгоритмы организуются при помощи оператора цикла **WHILE**.

Для вызова из одной процедуры других используется оператор **CALL**. Допустимы рекурсивные вызовы процедур.

Процедуры могут получать входные параметры и возвращать результат работы через механизм возвращаемого значения (оператор **RETURN**) и/или выходные параметры. Результатом работы процедуры может являться не только скалярное значение, но и курсор (выборка).

Для обработки результатов SELECT-запросов в процедурах используются курсоры (CURSOR), тип которых объявляется в соответствии со структурой ответа. Цикл работы с курсором может включать его открытие оператором OPEN (как результат запроса или выполнения другой процедуры), выборку данных оператором FETCH (в любом направлении) и закрытие (CLOSE) или, если процедура возвращает курсор, возврат (RETURN).

Процедуры могут работать со столбцами типа BLOB. Для этого используются стандартные функции чтения/записи в BLOB, который ассоциируется с текущей строкой курсора.

Понятие «курсor» используется исключительно для выборки данных. Для выполнения любых DML и DDL запросов (запросов отличных от SELECT-запроса) используется оператор EXECUTE.

Все операции процедур по модификации данных входят в пользовательскую транзакцию. Завершением транзакции управляет пользователь, однако, процедура также может зафиксировать или откатить изменения, сделанные в ее теле (и теле ее дочерних процедур) операторами COMMIT и ROLLBACK.

Для упрощения обработки ошибок в языке хранимых процедур предусмотрен механизм работы с исключительными ситуациями, в качестве которых могут рассматриваться ошибки выполнения SQL – запросов, ошибки времени исполнения (вызов несуществующей процедуры, деление на ноль и т.д.) или пользовательские исключения.

В момент возникновения исключения управление сразу автоматически передается на соответствующую ветку блока обработки исключений (EXCEPTIONS), что избавляет от необходимости «засорения» кода процедуры многочисленными условными операторами, проверяющими результат завершения каждого оператора. Процедура может обработать исключение, либо завершиться и передать исключение на верхний уровень (оператор RESIGNAL).

Для организации работы с хранимыми процедурами и триггерами СУБД ЛИНТЕР содержит транслятор хранимых процедур, преобразовывающий исходный текст процедуры/триггера в оттранслированный код, позволяющий быстро выполнять процедуру в исполняющей подсистеме.

Этот код хранится в специальной системной таблице \$\$\$PROC, описания входных/выходных параметров процедур хранятся в таблице параметров \$\$\$PRCD. Обе эти таблицы должны быть созданы перед началом работы с процедурами/триггерами (для триггеров так же необходимо наличие таблицы \$\$\$TRIG).

В поле типа BLOB таблицы \$\$\$PROC сохраняются оттранслированный код и исходный текст процедуры или триггера (последнее позволяет пользователю получать исходный текст в целях редактирования). Процедуры и триггеры создаются с помощью соответствующих запросов SQL типа CREATE/ALTER PROCEDURE, CREATE TRIGGER.

В случае ошибки в тексте процедуры/триггера возвращается специальный код ошибки ЛИНТЕР и интерактивные утилиты выдают подробную расшифровку в какой строке и какая ошибка произошла. В этом случае в таблице \$\$\$PROC создается только одна запись, содержащая исходный текст, что в последствии позволяет пользователю получить его и исправить ошибки.

Возможно использование хранимых процедур как функций, расширяющих язык SQL.

Для запуска процедур используется запрос

EXECUTE <процедура>(<параметры>)

в котором указывается имя и параметры процедуры. При передаче параметров можно использовать механизм передачи значений по умолчанию, когда указывается не весь список, а лишь интересующие значения.

С. Триггеры: связь событий и хранимых процедур

Хранимые процедуры очень удобны, когда пользователь хочет реализовать некоторый достаточно сложный алгоритм обработки данных и запустить его в нужный момент. Однако в приложениях часто требуется выполнить какие-либо действия при условии возникновения определенной ситуации.

Триггеры обеспечивают целостность, выполняя комплексную межтабличную проверку допустимости данных вне контекста справочной целостности и ограничений проверки.

Наиболее важное назначение триггеров – определение глобального делового правила. Например, триггер может использоваться для обеспечения того, чтобы менеджеру службы продаж посылалось сообщение о недостаточном количестве товара на складе и необходимости пополнения запаса, если общее количество товара на складе стало меньше чем зарезервированное.

Другой пример – при добавлении новых строк в одну таблицу в целях обеспечения целостности данных может потребоваться соответствующим образом добавлять или изменять строки других, связанных, таблиц.

Итак, **триггер** – это хранимая процедура, которая автоматически вызывается при выполнении того или иного действия с конкретной таблицей, направленного на изменение данных. Явно триггер вызываться не может. При создании триггера указывается, к какой таблице и к какому действию (UPDATE, INSERT или DELETE) он привязывается.

Триггер может вызываться один раз при выполнении всего запроса, либо каждый раз, когда необходимо обновить, удалить или вставить очередную строку в таблице (триггер FOR EACH ROW). Причем можно указать, когда должен вызываться триггер: до операции (триггер BEFORE) или после (триггер AFTER).

Основной частью триггера является его тело, которое определяется точно так же, как тело хранимой процедуры и содержит *алгоритм реакции триггера* на вызвавшее его действие. В теле триггера допустимы любые SQL-запросы, вызовы хранимых процедур и т.д. Для триггеров на каждую строку внутри тела можно использовать значения полей записи, которые были до операции изменения и значения, которые должны быть установлены после операции.

Эти значения доступны через поля предопределенных переменных-курсоров OLD и NEW, структура которых соответствует структуре таблицы. В триггере BEFORE можно присвоить полям переменной NEW новые значения, и при выполнении операции INSERT или UPDATE они будут использованы вместо старых.

Единственным значением, которое может вернуть триггер, является логическое значение, при чем, если триггер BEFORE возвращает «ложно», соответствующая операция отменяется для данной строки или всего SQL-оператора, в зависимости от вида триггера.

Д. Отладка хранимых процедур ЛИНТЕР®

Для упрощения процесса разработки хранимых процедур и триггеров ядро ЛИНТЕР содержит встроенный механизм отладки хранимых процедур. Пользователь отлаживает процедуру (триггер) при помощи специальной утилиты – отладчика,

который обменивается с ядром специальными управляющими командами. Наличие встроенного в ядро механизма отладки позволяет отлаживать процедуры (триггеры) *в реальных рабочих условиях (на сервере), в среде запросов и действий конкретного реального приложения.*

Для того чтобы начать отладку, пользователь открывает так называемую «отладочную сессию» для процедуры, после чего ядро работает с данной процедурой (триггером) в режиме отладки, реагируя на команды отладчика и передавая ему всю необходимую информацию.

Можно отлаживать лишь те процедуры (триггеры), которые оттранслированы с отладочной информацией. На необходимость включения отладочной информации указывает ключевая фраза **FOR DEBUG** в заголовке процедуры или триггера, как в приведенных выше примерах.

Для начала отладки конкретной исполняемой процедуры (триггера) необходимо либо запустить процедуру из-под отладчика, либо ждать, пока какое-нибудь приложение не вызовет эту процедуру или какой-нибудь запрос не запустит триггер (для триггеров это единственный способ).

Второй вариант предоставляет мощную возможность по отладке процедур и триггеров в рамках их «родной» среды, когда они запущены в штатном порядке. Отладчик позволяет просматривать исходный текст процедуры (триггера), выделяя строку с текущим исполняемым оператором.

Процесс отладки включает возможность пошагового выполнения процедур (триггеров), установки разнообразных точек останова и выполнения до одной из них, выполнения с заходом внутрь вызываемых дочерних процедур, наблюдения за значениями локальных переменных и конкретных выражений, стеком вызова.

Кроме обычных точек останова (останов каждый раз при выполнении конкретного оператора) поддерживаются так же условные точки (останов только в случае выполнения конкретного условия, например, $i > 10$) и точки останова на изменение значений.

Отладчик также помогает разобраться в причине возникновения исключений, приостанавливая выполнение процедур и указывая место, где произошло исключение.

VI. Надежность системы

A. Системный журнал

Основой надежности работы СУБД ЛИНТЕР с данными является *системный журнал* или *журнал транзакций*. В файле журнала отображаются все изменения, производимые над данными всеми пользователями системы.

Переход к *многоверсионности данных*, позволил «накрыть» все уровни изолированности, описанные в стандарте **SQL92 Entry – Entry Level** спецификация Международной Организации по Стандартизации (ISO) 9075-1992 языка баз данных SQL или сокращённо **SQL92E**.

Таким образом, сбои оборудования/питания не могут привести к потере или порче данных. При следующем старте система просканирует журнал и базу в поисках возможных несоответствий. Все некорректные данные базы будут исправлены, и только после этого система начнет обрабатывать запросы прикладных задач.

При исправлении *последних* изменений СУБД может столкнуться с тем, что часть из них достигла дискового пространства файлов полностью, некоторые изменения лишь частично записались на диск, а некоторые так и остались в буферах ядра системы, не затронув собственно файлы базы данных.

Итак, ЛИНТЕР при исправлении неполных/неверных данных может либо вернуть их в то состояние, в котором они находились до изменений (*откат изменений*), либо продолжить модификацию данных, следуя журналу транзакций (*прокрутка вперёд*).

При этом работает следующее простое правило: если пользователь получил «уведомление» о том, что его изменения перенесены в базу, то при следующем запуске системы новые данные обязаны находиться в базе.

1. Режимы транзакций СУБД ЛИНТЕР®

В ЛИНТЕР реализованы *три основных стратегии* (или режима) работы транзакций.

- **Оптимистичная (Optimistic Concurrency Control).**

Модификации данных, сделанные внутри одной транзакции, не поступают в базу, однако, поступают в системный журнал вместе со старым образом данных. При фиксации изменений транзакции система начинает перенос изменений из журнала в базу данных.

При переносе очередного изменения ЛИНТЕР сравнивает старый образ модифицируемых данных из журнала и из базы. Если образы совпадают (данные никто не менял), то это очередное изменение попадает в базу, в противном случае (кто-то уже успел раньше изменить и зафиксировать данные) откатывается вся транзакция, а приложению возвращается соответствующее сообщение.

Этот режим работы транзакций при всех его недостатках обладает существенным достоинством – скоростью. Данные не блокируются, каждая задача работает исходя из оптимистичного предположения о том, что кроме неё с базой не работает никто, и никто никого не ждёт. А при возникшей коллизии транзакцию можно и повторить. Что очень *удобно особенно в системах массового обслуживания*.

- **Пессимистичная (Pessimistic).**

Это классический режим работы транзакций. Все изменения поступают сразу в базу данных, а также в системный журнал. Однако эти изменения не видны другим

транзакциям до тех пор, пока не пройдет её фиксация. До тех пор работает аппарат блокировок, запрещающий модификацию читаемых или чтение модифицируемых данных.

- **Стратегия read-only.**

Удобный режим транзакций, которые не делают никаких изменений в базе данных, их функция – только чтение (например, отчёты за день, сведение данных для анализа и т.п.). Обычно такие транзакции вовлекают в свою работу большие объёмы данных, так что блокировка всех используемых данных может привести к длительной задержке прочих транзакций.

В ЛИНТЕР транзакция read-only *не блокирует данные*. Но при её появлении, все другие транзакции, изменяя данные, оставляют (для неё и ей подобных) тот образ, который имели данные, когда транзакция read-only стартовала.

Таким образом, транзакция read-only получает базу данных именно на момент её старта, то есть как бы моментальный «снимок» базы данных.

В ЛИНТЕР 6.1 реализовано четыре уровня изоляции транзакций: READ UNCOMMITTED(1), READ COMMITTED(2), SERIALIZABLE(3), OPTIMISTIC(4) (OPTIMISTIC оставлен для совместимости с предыдущими версиями СУБД). Следует отметить, что в отличие от предыдущих версий, выборка стабильна при любых уровнях изолированности транзакций.

READ_UNCOMMITTED

При работе в этом режиме транзакция видит все свои изменения и изменения, сделанные транзакциями уровней 1-3 (фиксированные и нефиксированные). Транзакция не видит нефиксированных изменений, внесенных OPTIMISTIC транзакцией. Выборка стабильна.

Транзакция не может менять данные, измененные и не фиксированные другими транзакциями. При попытке сделать это запрос будет либо ждать завершения конкурирующей транзакции, либо выдаст сообщение об ошибке, говорящее, что данные были изменены (в зависимости от того, какой режим работы канала выбран – блокирующий или неблокирующий).

При попытке создать/модифицировать/удалить запись так, что происходит конфликт ссылочной целостности с нефиксированными данными, сделанными другими транзакциями, запрос будет либо ждать завершения конкурирующей транзакции, либо выдаст сообщение об ошибке, говорящее, что данные были изменены.

При попытке сделать ссылку на нефиксированные данные (FK) запрос будет ждать завершения конкурирующей транзакции или завершится ошибкой.

READ_COMMITTED

Транзакция видит все свои изменения и фиксированные изменения, сделанные транзакциями уровней 1-4. Выборка стабильна.

Транзакция не может изменять данные, измененные и нефиксированные другими транзакциями. При попытке сделать это, запрос будет либо ждать завершения конкурирующей транзакции, либо выдаст сообщение об ошибке, говорящее, что данные были изменены (в зависимости от того, какой режим работы канала выбран – блокирующий или неблокирующий).

При попытке создать/модифицировать/удалить запись так, что происходит конфликт ссылочной целостности с нефиксированными данными, сделанными другими

транзакциями, запрос будет либо ждать завершения конкурирующей транзакции, либо выдаст сообщение об ошибке, говорящее о конфликте по ссылочной целостности.

SERIALIZABLE

Транзакция видит все свои изменения и фиксированные изменения, сделанные транзакциями уровней 1-4, которые завершились до ее старта. Выборка стабильна.

Транзакция не может менять данные, измененные и не фиксированные другими транзакциями. При попытке сделать это запрос будет либо ждать завершения конкурирующей транзакции, либо выдаст сообщение об ошибке.

При попытке создать/модифицировать/удалить запись так, что происходит конфликт ссылочной целостности с нефиксированными данными, сделанными другими транзакциями, запрос будет либо ждать завершения конкурирующей транзакции, либо выдаст сообщение о конфликте по ссылочной целостности.

При попытке модифицировать данные, которые были модифицированы (удалены), причем эти изменения были фиксированы после ее старта, запрос завершится ошибкой **НЕСЕРИЙНЫЙ ДОСТУП К ДАННЫМ (8102)**.

Одновременно может существовать ограниченное число соединений с режимом SERIALIZABLE (100). Попытка создать большее число соединений с режимом SERIALIZABLE приведет к сообщению об ошибке **"НЕТ СВОБОДНОЙ ТОЧКИ ВХОДА В ТАБЛИЦЕ СЕРИЙНЫХ ТРАНЗАКЦИЙ (8107)"**.

2. Иерархия транзакций в СУБД ЛИНТЕР®

В СУБД ЛИНТЕР каждое приложение может использовать как обычные *плоские* транзакции, так и *иерархические*.

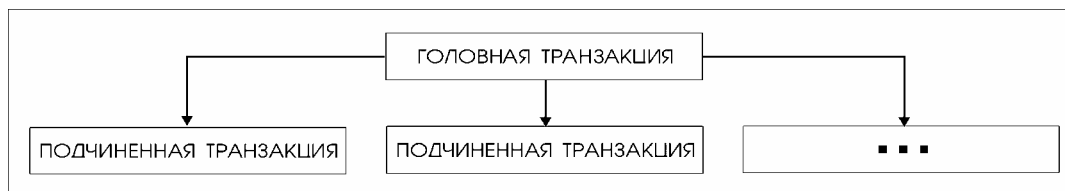
Классические *плоские* транзакции представляют собой последовательность запросов, завершающуюся оператором фиксации/отката Commit/Rollback. Причём необходимость отката транзакции может возникнуть и в середине транзакции. При этом из базы данных будет удалены *все* изменения, которые успела сделать транзакция. Приложение не может откатить лишь отдельные изменения, ему разрешено руководствоваться только одним принципом – «или всё, или ничего».

Несомненно, это слишком категоричный и, следовательно, не всегда приемлемый принцип. Особенно в системах с нечёткой логикой.

Например, приложению (см. следующий рисунок) нужно произвести три действия **A**, **B** и **D** в одной транзакции. Причём действия **A** и **D** являются необходимыми, а вот для действия **B** есть «запасной вариант» – действие **C**. Так что приложение после выполнения действия **A** пытается в первую очередь выполнить действие **B**, и только если по какой-либо причине результаты обработки **B** «не устраивают» или возникла ошибка, то изменения, сделанные **B**, откатываются, и приложение предпринимает резервное действие **C**.



Такую логику транзакции не удастся реализовать при помощи линейных транзакций. Для использования подобных транзакций СУБД ЛИНТЕР предоставляет пользователю возможность иерархии транзакций.



При этом действуют следующие правила.

1. Подчинённые (*вложенные*) транзакции действуют по линейному принципу. И могут работать независимо друг от друга в любом из режимов (от оптимистичного до read-only).
2. Каждая из подчинённых транзакций может быть фиксирована/откачена независимо от других подчинённых транзакций.
3. Подчинённые транзакции «видят» изменения, сделанные прочими подчинёнными транзакциями (имеющими общего предка). Полностью изолированы только транзакции различных иерархий.
4. Фиксация/откат головной транзакции приведёт к фиксации/откату всех подчинённых транзакций.

Подобный подход к системе использования транзакций гораздо более гибкий.

Кроме того, в СУБД ЛИНТЕР дополнительно введён аппарат контрольных точек (SAVE POINTS). При помощи этого аппарата можно будет откатить все изменения (даже фиксированные) до указанной контрольной точки.

В. Архивирование таблиц (базы данных)

Утилита архивирования ЛИНТЕР – **lhb** позволяет сохранять (со сжатием) базы данных целиком/выборочно.

При этом работа **lhb** не останавливает работу других пользователей СУБД. **lhb** делает просто «сжатый снимок» базы данных на какой-то момент времени.

Таким образом, поврежденная база данных может быть всегда восстановлена из последнего **lhb**-архива.

lhb – мощная программа, обладающая возможностями, которые облегчат пользователю организацию необходимого алгоритма архивирования.

Здесь искушенный пользователь обнаружит следующие возможности:

- ведение архива (системы архивов) на магнитной ленте;
- нарастающее (инкрементное) архивирование;
- установка системы и расписания (графика) архивирования базы данных;
- развитая система реакций на временные события;
- возможность написания скриптов на языке архиватора **BSL** (Backup Script Language);
- мощная система для селекции архивируемых объектов базы данных;
- тестирование архива;
- шифрация данных архива и авторизация доступа к нему.

Особый интерес, конечно же, представляет определение графика или сценария архивирования с помощью языка BSL, наиболее важными возможностями которого являются:

- отслеживание интервалов времени для запуска **lhb** или других программ в нужный момент,
- достаточно развитая система управления работы программы (операторы типа if, else, while, ...),
- система обработки ошибок, возникающих при запуске программ,
- функции для управления файлами (копирование, перенос, удаление и пр.),
- набор операций со строковыми переменными.

Сам сценарий или график на языке BSL составляет пользователь (с правами администратора). Здесь могут быть указаны периодичность и порядок автоматических действий архиватора, например:

- архивирование базы данных;
- создание резервных копий;
- перенос архива или резервных копий на другие носители данных (в том числе и на магнитную ленту);
- удаление устаревших архивов и их копий.

Файл сценария может содержать определение периодичности действий (ежедневно + время, еженедельно + день недели + время, ежемесячно + число месяца + время), а также особенные даты, выходящие за пределы периодичности (когда требуется или, наоборот, исключено действие). Например:

Variables:

NUM = 1;

Rights:

```
Everyday ( time = '12:00' ) {
if ( CWEEKDAY() == "Sun" ) { /* New archive on Sunday */
move (FILENAME + TOSTR(NUM) + ".lhb", "c:\arc");
NUM = NUM + 1;
backup ( "s -u " + NAME + "/" + PASSWORD + " -f " + FILENAME + TOSTR(NUM) + ".lhb -startinc" );
}
else /* Incremental backup */
backup ( "s -u " + NAME + "/" + PASSWORD + " -f " + FILENAME + TOSTR(NUM) + ".lhb -inc" );
Exception: /* for 'everyday' */
print("Error=" + TOSTR(CERROR) + ", LinError=" + TOSTR(LINERROR) + ", SysError=" + TOSTR(SYSERROR) );
stop;
}
```

Special:

```
before { /* just after the start */
print ( "Start backup system" );
backup("s -u " + NAME + "/" + PASSWORD + " -f " + FILENAME + TOSTR(NUM) + ".lhb -startinc" );
}
after { /* after 'stop' or Ctrl-C */
print ( "Stop backup system" );
if ( ERROR != 0 ) logprint ( "Error present: " + TOSTR(ERROR) );
}
iferr { /* global */
print("Error=" + TOSTR(CERROR) + ", LinError=" + TOSTR(LINERROR) + ", SysError=" + TOSTR(SYSERROR) );
stop;
}
```

С. Горячее резервирование

Подсистема горячего резервирования ЛИНТЕР предназначена для повышения надежности информационных систем за счет полного дублирования процесса ведения базы данных на резервном сервере базы данных.

В момент аварии одного из задублированных серверов процесс взаимодействия с базой данных переключается на второй. Горячее резервирование означает, что выход из строя одного из дублирующих серверов не будет замечен прикладными процессами, имеющими выполняющиеся запросы к базе данных, и не повлияет на результаты последующих запросов.

Процесс резервирования «прозрачен» для приложений, так как способ их доступа к базе не зависит от того, выполняется оно или нет.

Подсистема резервирования реализована в виде двух процессов. Один из них работает на узлах процессов клиентов, другой – на каждом из задублированных серверов.

Переход системы из дублирующего режима в монопольный происходит в следующих случаях:

- сервер базы данных был остановлен оператором;
- один из серверов базы данных выключился или «завис»;
- на одном из серверов базы данных произошла неисправимая системная ошибка;
- «главный» сервер базы данных обнаружил несовпадение результатов обработки запросов с резервным сервером.

Более того, система ЛИНТЕР позволит «поднять» упавший сервер, не останавливая работу всей системы. После старта резервный сервер «догонит» уже находящийся в работе, и только после этого, система снова перейдет в режим горячего резервирования.

D. Testdb – утилита проверки физических структур базы

Утилита Testdb поможет администратору обнаружить возможные ошибки, появившиеся в базе данных в процессе работы системы.

Testdb может работать как со всей базой данных, так и выборочно обследовать указанные таблицы. При этом может быть указана ещё и глубина, а также сложность проверок базы.

Если были диагностированы ошибки, то Testdb попытается их исправить собственными силами или предложит пользователю «лечащий» пакет SQL запросов.

На данный момент Testdb – программа, которая может работать с базой только монопольно. Никакой другой процесс не может работать с файлами базы данных одновременно с ней, в том числе и ядро СУБД.

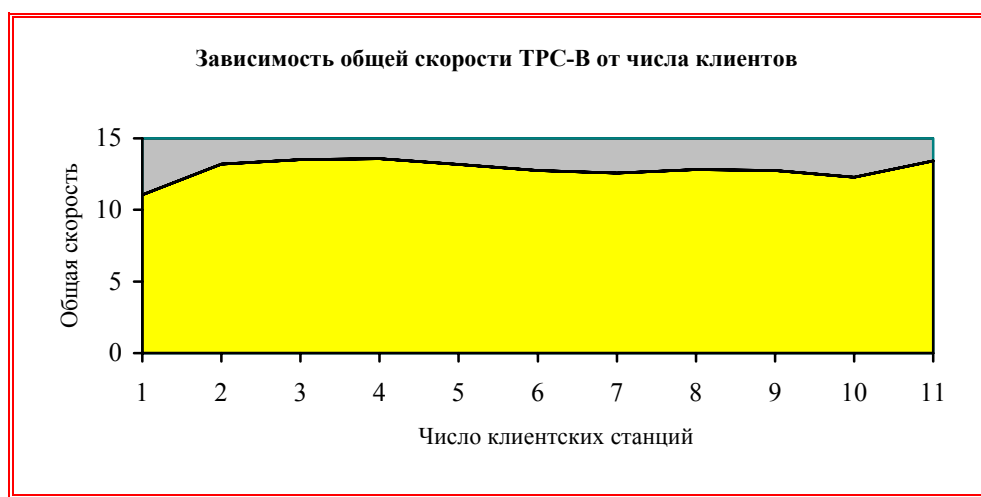
Однако зачастую требуется проверка (и, может быть, «лечение») базы данных как раз во время работы системы (например, в непрерывных производствах). Для этого в систему введён аппарат «горячего» тестирования таблицы. Таким образом, любое приложение может подать SQL-запрос на тестирование таблицы и получить информацию об ошибках, которые необходимо исправить. Тестирование может иметь самый низший приоритет и, следовательно, выполняться в фоновом режиме, т.е. только тогда, когда нет других, более приоритетных запросов.

VII. Производительность

Существующие промышленные приложения на базе ЛИНТЕР показали, что система обладает достаточной производительностью для решения любых информационных задач и автоматизированных систем (от управления производством до управления технологическими процессами – плавильные печи, городское энергоснабжение, диспетчерская служба аэропорта по управлению самолётами, контроль атомных реакторов и т.д.).

Кроме того, ЛИНТЕР – одна из немногих систем, обладающих минимальными требованиями к вычислительным ресурсам. Даже на малых и средних серверах ЛИНТЕР покажет приемлемое быстродействие.

Наш опыт показывает, что даже средний Pentium (2xP5 233, 128Mb, RAID) эффективно поддержит около 50-ти клиентских приложений. Это подтверждают и показания стандартных тестов, например, TPC-B (Pentium 2xP5 233, 128Mb, RAID; Windows NT), см. диаграмму, приведённую ниже:



VIII. Масштабируемость

Масштабируемость СУБД ЛИНТЕР, т.е. зависимость эффективности её работы от вычислительных ресурсов, это достаточно многомерное свойство. Можно рассматривать масштабируемость по отношению к объёму оперативной памяти, числу процессоров на сервере и т.п.

1. Почти все процессы, участвующие в обработке запросов на сервере (собственно ядро ЛИНТЕР, SQL-транслятор, транслятор хранимых процедур, сортировка, сетевой драйвер сервера), настраиваются на использование дополнительной памяти.

Использование дополнительной памяти этими компонентами серьёзно увеличивает эффективность работы системы.

На приведённой ниже диаграмме заметно, что скорость работы однопользовательского теста ТРС-В пропорционально зависит от размеров занимаемой системой (под пул страниц) оперативной памяти.

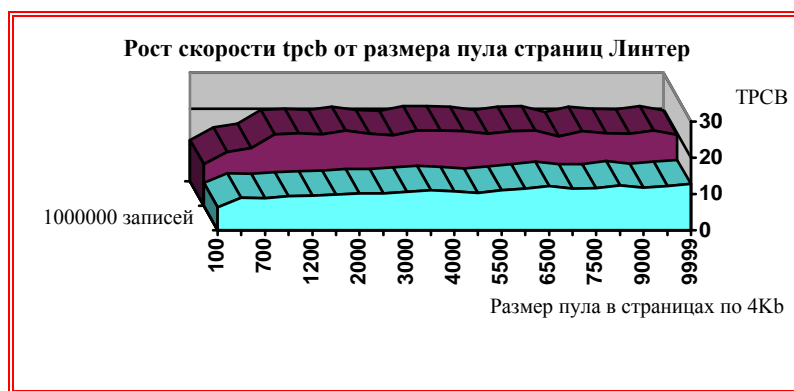


Диаграмма ТРС-В на 100.000 записей прекращает свой рост, начиная с пула в 1.000 страниц, так как база данных вся поместилась в ОЗУ, и дальнейшее увеличение пула не даёт роста эффективности.

График ТРС-В на 1.000.000 записей представляет собой почти пропорциональную зависимость – прирост эффективности ~3% на каждые дополнительные 1.000 страниц оперативной памяти.

2. Зависимость эффективности от числа процессоров сервера не столь яркая. ЛИНТЕР – многозадачная СУБД. В процессе обработки запросов принимает участие несколько задач: ядро, SQL-транслятор, транслятор хранимых процедур и процессор сортировки (может быть, даже не один).

При этом всё может быть организовано таким образом, что различные из указанных задач будут выполняться на различных процессорах. И это даёт определённый выигрыш в эффективности.

В СУБД ЛИНТЕР 6.2 осуществлён переход к использованию различных процессоров при обработке запросов внутри ядра. Причём принят самый интересный вариант – выполняться на различных процессорах смогут не только различные запросы, но даже и различные части одного запроса.

3. При наличии высококачественных интерфейсов ввода-вывода (например, SCSI, RAID-массивы) ЛИНТЕР будет использовать их возможности по параллельному исполнению нескольких запросов на ввод-вывод.

IX. Совместимость

Первым признаком совместимости можно считать язык запросов SQL ЛИНТЕР. Уже то, что этот язык основан на международном стандарте (см. гл. V «Внутренние языковые средства (SQL, хранимые процедуры, триггеры)»), говорит о высокой совместимости системы.

Кроме того, в SQL ЛИНТЕР введены многие элементы, обеспечивающие совместимость с другими системами, главным образом с СУБД Oracle.

Псевдографические интерфейсные утилиты системы ориентированы на стандарт CUA. Пользователь, привыкший к аналогичным системам, легко освоит работу интерфейса и в среде ЛИНТЕР.

Интерфейс ODBC позволяет осуществлять максимальную переносимость приложения с одной СУБД на другую без учета их специфики. Поэтому в связи с совместимостью, конечно же, необходимо упомянуть об ODBC-драйвере ЛИНТЕР (спецификация – ODBC 3.x), а это значит, что все существующие средства разработки и то, что опирается на ODBC-стандарт, будут работать с СУБД ЛИНТЕР без ограничений.

Интерфейс OLE DB позволяет осуществлять доступ к данным как напрямую, так и посредством ADO в среде Windows.

Наличие драйвера dbExpress в СУБД ЛИНТЕР обеспечивает поддержку разработки приложений в среде последних версий Delphi и Kylix.

До появления Kylix пакет Delphi для Windows предлагал несколько механизмов доступа к базам данных – ADO, Borland Database Engine (BDE) и InterBase Express. Однако в среде Linux ни один из этих механизмов недоступен, поэтому Borland разработала новую кросс-платформенную технологию доступа к данным – dbExpress, работающую как в Windows (Delphi), так и в Linux (Kylix).

ЛИНТЕР JDBC (спецификация JDBC 1.0, 2.0, 3.0) позволяет писать приложения на Java, используя СУБД ЛИНТЕР.

Java-программа может быть разработана в виде апплета, загружаемого через Internet и запускаемого на стороне клиента, или в виде приложения, постоянно находящегося на стороне клиента. В любом случае интерфейс JDBC позволяет Java-приложению подключаться к удаленным базам данных, направлять к ним запросы и получать результаты обработки запросов.

При этом реализованы следующие типы драйверов

- **Pure Java client** – клиент полностью написан на Java. Налицо преимущества этого типа – реализация классов полностью на Java позволяет отказаться от дополнительных компонент на клиентской ЭВМ. К недостаткам этого типа можно отнести относительно невысокую производительность из-за медлительности **java.net.***.
- **Part-Java client** – кроме классов, реализующих интерфейс JDBC, для работы необходимо иметь динамически подключаемую библиотеку трансляции вызовов к СУБД ЛИНТЕР. К недостаткам этого типа можно отнести необходимость инсталляции этой библиотеки, зато это даст существенный выигрыш в скорости (в 3-5 раз по сравнению с использованием драйвера типа **Pure Java client**).

Основные сетевые протоколы, поддерживаемые СУБД ЛИНТЕР: TCP/IP, IPX/SPX, DECNet.

Помимо перечисленного, в системе реализованы средства конвертации данных из DBF-формата и программные интерфейсы для работы с ЛИНТЕР из Clipper, FoxPro.

Отдельно следует отметить совместимость ЛИНТЕР с СУБД Oracle на уровне **Pro*С** и **OCI**. Благодаря библиотеке **OraLin**, приложение, работающее с СУБД Oracle, будет с таким же успехом работать (после перекомпоновки, а в Windows и без неё, в случае использования динамически подгружаемых библиотек) и с ЛИНТЕР, совершенно не заметив подмены.

Кроме того, для совместимости с СУБД Oracle был расширен синтаксис языка запросов, который в данный момент поддерживает многое из того, что и **Oracle SQL**, вплоть до иерархических запросов. Для многочисленных пользователей Oracle переход к использованию СУБД ЛИНТЕР не составит большого труда.

В ЛИНТЕР также реализовано средство – **LinTcl**, которое расширяет возможности **Tcl/Tk** (Tcl – Tool Command Language, Tk – Tool kit) для работы с СУБД ЛИНТЕР, что позволяет использовать Tcl/Tk для быстрого написания информационных приложений в операционных средах, в которых работает Tcl/Tk (Unix-подобные системы, Windows).

Х. ЛИНТЕР® в сетевых условиях

За работу ЛИНТЕР в сетевых условиях отвечают две компоненты:

- *сетевой драйвер клиента,*
- *сетевой драйвер сервера.*

К этим компонентам предъявляются те же требования надежности, что и ко всей системе. Так сетевые компоненты отслеживают правильность структуры и некоторые контрольные суммы всех сообщений, проходящих через них, а также «живучесть» всех объектов, связанных с ними и т.п.

Кроме того, сетевые компоненты обладают чрезвычайно скромными требованиями к вычислительным ресурсам. Например, драйверу сервера необходим ~1Кб на каждое соединение.

Основа работы в сети – файл **nodetab**. В этом файле должны присутствовать все необходимые ЛИНТЕР-серверы: их логические имена, адреса, протоколы, операционные системы и т.п.

А. Режим «клиент-сервер»

Если приложениям не требуется обращаться за данными к двум серверам одновременно (когда все они работают только с одним сервером), то такой режим работы называется «клиент-сервер».

Для этого режима можно не указывать, на какой именно сервер (из тех, что в наличии) обращается приложение. Драйвер клиента можно настроить на работу с этим сервером, как с сервером по умолчанию.

В этом режиме драйвер клиента исполняет для приложения роль СУБД. Поэтому приложение можно вообще отлаживать в локальном режиме (на одной машине, без использования сети), и уже потом без изменений использовать его в сетевом режиме «клиент-сервер».

В. Режим «мультисервер»

Если приложению необходимо использовать несколько серверов баз данных (например, для перекачки информации с одного сервера на другой), то ему придётся явно указывать, к какому из них он обращается в каждый конкретный момент.

При этом используются *логические имена* серверов (описанных в файле **nodetab**). Эти имена передаются вместе с запросом и необходимы драйверу клиента для точной адресации при посылке запроса к нужному серверу.

Такой режим работы в документации по системе ЛИНТЕР называется «мультисервер». Отличие его от распределённости в том, что базы данных, расположенные на различных серверах, не составляют единого целого, таблицы различных серверов не могут участвовать в одном запросе, а также в том, что приложению необходимо явно указывать те серверы, к которым оно обращается.

Отлаживать такое приложение можно только в режиме «мультисервер».

С. *Распределенная база данных*

Концепция распределённости СУБД ЛИНТЕР позволяет прозрачно обрабатывать запросы к данным, находящимся в различных базах данных вне зависимости от их физического расположения. Это позволяет базам данных функционировать независимо друг от друга. Кроме того, эта концепция обеспечивает равноправный доступ пользователей к разным базам данных, расположенным в различных узлах вычислительной сети.

Основным понятием концепции является *сетевое или логическое ЛИНТЕР-имя*, которое представляет собой восьмисимвольное имя, прописанное в специальном файле ЛИНТЕР-имён – **nodetab**. В этом файле сетевые ЛИНТЕР-имена связаны с сетевыми параметрами узла запуска ядра ЛИНТЕР.

Кроме того, сетевое ЛИНТЕР-имя однозначно определяет в текущей операционной среде базу данных, для которой будет запущено независимое ядро СУБД ЛИНТЕР. Части выполняемого запроса, которые необходимо выполнить именно в данной базе будут переправляться по указанному в **nodetab** адресу с использованием ЛИНТЕР-имени.

Список баз данных (ЛИНТЕР-имён), которые могут участвовать в процессе выполнения распределённых запросов, содержится в специальной системной таблице – **SERVERS**. Эта таблица входит в системный словарь базы данных и подчиняется стандартным правилам работы со словарями, принятым в СУБД ЛИНТЕР.

В **SERVERS** хранится имя, по которому работает СУБД внутри и которое будет передано вовне для определения, на какой из удалённых серверов нужно пересылать запрос за недостающими данными.

В **SERVERS** не хранится связь имени с сетевой конфигурацией. Таким образом, при изменениях в сетевой конфигурации (через файл **nodetab**) трансляция имен внутри базы данных останется неизменной.

Создание/удаление узла распределённой базы данных производится специальным SQL-запросом CREATE/DROP NODE.

Теперь о технологии распределённости в СУБД ЛИНТЕР.

Главная компонента аппарата распределённости – **loltp** – процесс, который работает с удалённым сервером. Он принимает запрос от локального ядра во внутреннем формате и формирует из него SQL-запрос к удалённому серверу. Соответственно, при этом используется сетевой клиентский драйвер.

Другая модификация системы позволяет создавать многомашинные СУБД-комплексы на основе асинхронной репликации.

Такие системы используются чаще всего в WEB-приложениях, где запросы на поиск гораздо более часты, нежели запросы на модификацию (поисковые системы, интернет-магазины и т.п.). В таких приложениях можно достичь практически пропорционального масштабирования при увеличении числа машин в комплексе, например, при двукратном увеличении числа ЭВМ, мы получим практически двукратное увеличение скорости обработки общего потока запросов.

XI. Основные характеристики системы

СУБД ЛИНТЕР –

- открытая система;
- многопользовательская система;
- реляционная система

| | |
|---|----------------|
| Язык запросов | SQL |
| Максимальное число таблиц в запросе (на одном уровне) | 32 |
| Максимальное число таблиц в базе данных | 65.535 |
| Максимальное число столбцов в одной таблице | 250 |
| Максимальное число строк в таблице | 1 млрд. |
| Максимальная длина строки (в байтах) | 65.000 |
| Максимальное количество ключей в таблице | 250 |
| Максимальная длина ключа (в байтах) | 1024 |

СУБД ЛИНТЕР поддерживает типы данных, отраженные в следующей таблице

| Тип | Обозначение | Длина | Аналог в С |
|---|--------------------------|-----------------------------|------------------|
| Строка символов | CHAR | до 4.000 СИМВОЛОВ | массив байтов |
| Строка символов переменной длины | VARCHAR | до 4.000 СИМВОЛОВ | аналога нет |
| Строка байтов | BYTE | до 4.000 байт | массив байтов |
| Строка байтов переменной длины | VARBYTE | до 4.000 байт | аналога нет |
| Строка UNICODE символов | NCHAR | до 2000 СИМВОЛОВ | аналога нет |
| Строка UNICODE символов переменной длины | NCHAR VARYING | до 2000 СИМВОЛОВ | аналога нет |
| булевское | BOOLEAN | 1 байт | аналога нет |
| Короткое целое | SMALLINT | 2 байта | short |
| Длинное целое | INTEGER | 4 байта | long |
| 64-битное целое | BIGINT | 8 байт | аналога нет |
| Вещественное число (плав. точка) | REAL | 4 байта | float |
| Вещественное число (плав. точка, двойная точн.) | DOUBLE | 8 байт | double |
| Вещественное число (фиксированная точка) | NUMERIC | 16 байт | аналога нет |
| Дата (совмещенная со временем) | DATE | 16 байт | аналога нет |
| Большой файловый объект | BLOB | до 2 млрд. байт | аналога нет |

СУБД ЛИНТЕР поддерживает несколько типов индексов:

- классические индексы по В*-дереву;
- словные,
- фразовые.

Минимальный уровень блокирования данных

запись

Данные и индексы таблицы хранятся в *сжатом виде*, кроме того, есть эффективный механизм переиспользования табличного пространства.

Режимы обработки транзакций:

**Optimistic,
Autocommit,
Pessimistic,
Read-only**

ХII. Асинхронная репликация

Многие современные поисковые системы (например, электронные магазины, информационные порталы и т.п.) предъявляют очень высокие требования к скорости отработки поисковых запросов при условии одновременной работы большого количества клиентов. Кроме того, развиваясь, такие системы должны легко масштабироваться без ущерба для скоростных характеристик системы.

Один из способов удовлетворения этой потребности – реализация в СУБД механизма асинхронной репликации. Основная идея репликации заключается в том, что вместо одной базы данных, с которой должны работать все клиенты, создается несколько одинаковых (по крайней мере, частично) баз данных на разных машинах. Клиенты имеют доступ к некоторому распределяющему устройству (реализованному аппаратно или каким-либо программным методом), которое при появлении нового клиента оценивает загрузку каждого сервера и направляет клиента на наименее загруженный, с которым он (клиент) и будет работать до отсоединения.

Серверы баз данных связаны между собой, и все сделанные изменения пересылают друг другу с тем, чтобы привести реплицируемые объекты (таблицы базы данных) в полное соответствие. Поскольку репликация асинхронная, этот процесс происходит не сразу, а в течение некоторого времени. В этот период данные на разных серверах будут отличаться.

Такое построение позволяет значительно (в идеальном случае, прямо пропорционально количеству серверов) увеличить производительность системы и наращивать её по мере роста нагрузки (увеличения количества клиентов или размеров базы данных) простым прибавлением серверов в систему репликации.

А. Правила репликации

Для управления системой на логическом уровне в СУБД ЛИНТЕР используются правила репликации, которые создаются обычным SQL-запросом и представляют собой описание того, какие объекты, куда и каким образом реплицировать. В ЛИНТЕР создание правила репликации выглядит так:

```
CREATE REPLICATION RULE имя_правила
FOR [ имя_пользователя. ] имя_таблицы
[ TO имя_удаленной_таблицы ]
ON NODE имя_сервера
[ USER имя_пользователя ] [ PASSWORD 'пароль' ]
[ ENABLE / DISABLE ]
[ SYNC / ASYNC ]
[ IGNORE OLD VALUE / CHECK OLD VALUE / CORRECT NUMBERS ];
```

Сама репликация происходит по первичным ключам – каждая таблица, подлежащая репликации должна содержать первичный ключ, значение которого используется для идентификации (при удалении и изменении значений) в реплицируемых таблицах.

Есть три модели поведения системы при возникновении рассогласования между значениями записей в базах данных реплицируемых серверов. Если такая ситуация возникла во время выполнения операции (update, delete), можно предпринять следующие действия:

- **IGNORE OLD VALUE** – игнорировать несовпадение старого значения,

- **CHECK OLD VALUE** – обязательно проверить старое состояние и вернуть ошибку, если нет полного совпадения,
- **CORRECT NUMBERS** – если не совпадают числовые значения, сохранить разницу между старым и новым значением.

Неразрешимые коллизии могут возникнуть при одновременной вставке во взаимно реплицируемые таблицы одинакового значения первичного ключа.

В. Сервер репликации СУБД ЛИНТЕР

В системе асинхронной репликации участвуют два или более серверов, на каждом из которых работает ЛИНТЕР и два процесса репликации, In и Out (они представляют собой отдельные потоки в Windows или процессы в UNIX). Объектами репликации являются таблицы базы данных, список которых вместе с правилами и адресами рассылки хранится в БД.

Сервер репликации (СР) представляет собой специальный процесс, который получает данные об измененных данных от СУБД ЛИНТЕР и сохраняет эти данные в очередях репликации в хранилище данных репликации (ХДР), которое представляет собой соответствующим образом «урезанное» ядро. Оно же будет использоваться процессами In и Out для получения данных, подлежащих репликации и рассылке.

Функции компонентов:

ЛИНТЕР – основное ядро, работает независимо от остальных компонентов. Должно обеспечивать только одну дополнительную функцию: выдавать для СР измененные записи.

Сервер репликации – запускается отдельно и независимо от СУБД, он в свою очередь запускает ХДР, In и Out; формирует структуру данных в ХДР, запрашивает и получает данные от ЛИНТЕР, сохраняет их в ХДР, формирует очередь рассылки в ХДР.

Хранилище данных репликации – это ЛИНТЕР, который хранит данные для рассылки. К этим данным имеют доступ СР, процессы In, Out и мониторы.

Процесс In – получает от удаленного Out информацию об измененных записях, после получения информации о подтверждении транзакции выполняет транзакцию, отсылая код возврата отправителю. Получаемые данные хранятся в ХДР в виде приемной очереди.

Процесс Out – ожидает завершения транзакций (хранящихся в ХДР) и рассылает данные по назначению. Получает и заносит в ХДР коды возврата от удаленных серверов.

Элемент очереди рассылки включает в себя полную информацию о старом и новом состоянии записи, адрес назначения, номер канала, производящего операцию, номер транзакции и время операции. Эта информация заносится в таблицу очереди рассылки на сервере репликации. В качестве первичного ключа этой таблицы используется время операции.

Процесс **In** получает данные и помещает их в приемную очередь, структура которой похожа на структуру таблицы очереди рассылки. После этого он формирует ответ, уведомляющий отправителя о нормальном приеме. Одновременно (возможно, с контролем над загруженностью ЛИНТЕР) происходит собственно репликация, коды завершения сохраняются в таблице приемной очереди.

В качестве идентификатора кортежа используется первичный ключ (для очереди рассылки это OPER_DATE (дата операции, она уникальна), на приемной очереди

это уже не первичный ключ, там идентификация происходит по OPER_DATE и SERVER_SRC (передающий сервер)), описание которого передается от Out к In и сохраняется в таблицах сервера репликации.

Если один из процессов (ЛИНТЕР или CP) завершается некорректно, этот процесс стартует заново, восстанавливается и работа продолжается. Повторное прохождение одного и того же блока отслеживается с помощью времени операции (OPER_DATE).

В качестве протоколов проделанной работы используются эти же очереди с соответствующими кодами завершения и создаваемый компонентами log-файл.

При необходимости администратор системы может запустить процедуру очистки очередей сервера репликации, при этом будут удалены все уже реплицированные записи, возможно, до указанной администратором даты.

C. Анализ особенностей системы.

Как уже было замечено, асинхронная репликация может использоваться в системах, которые предусматривают большое количество поисковых запросов при относительно незначительных изменениях. Это не значит, что нельзя делать массированных изменений БД, однако эффективность репликации высоко динамичных баз данных оказывается невысокой и, как правило, не соответствует поставленным задачам.

Кроме того, важной особенностью системы является возможность временного рассогласования данных на разных серверах в том случае, если клиенты производят изменения.

D. Достоинства и недостатки

Итак, к достоинствам систем асинхронной репликации следует отнести:

- Хорошую масштабируемость (стремящуюся к прямо пропорциональной зависимости от количества серверов, участвующих в процессе репликации в случае отсутствия изменяющих запросов)
- Высокую скорость выполнения запросов: в идеальном случае, если количество одновременно работающих клиентов равно или меньше, чем серверов в системе репликации – достигается предельное значение быстродействия: один клиент – один компьютер.
- Хорошая отказоустойчивость: отказ одного или нескольких серверов не приведет к остановке всей системы, а лишь немного замедлит работу, так как клиенты временно будут перераспределены между оставшимися серверами. Отказавший сервер может быть запущен в любой момент и сам произведет все необходимые действия для синхронизации с остальными.

Недостатки:

- Падение эффективности в случае высокой динамики изменений – рассылка и параллельные изменения всех БД снижают скорость отработки поисковых запросов.
- Временное рассогласование данных на серверах, которое практически исключает применение систем асинхронной репликации в приложениях, требующих абсолютной синхронности данных, получаемых разными клиентами.
- Необходимость нетривиального администрирования, разрешение коллизий с одинаковыми первичными ключами или по какой-либо причине

рассогласованными данными. Уменьшить вероятность неразрешимых коллизий (или даже исключить ее) можно на этапе проектирования приложения или, в ряде случаев, при создании самих баз данных на разных серверах (например, выделением для автоинкрементальных полей отдельных непересекающихся для каждого сервера диапазонов значений).

Е. Направление развития

Анализируя недостатки механизма асинхронной репликации можно предложить несколько усовершенствований, которые могли бы увеличить эффективность работы системы (большинство из них в настоящее время находятся на стадии реализации и войдут в следующие версии системы):

- Падение производительности во время проведения изменений подсказывает, что репликацию данных следует производить не во время их поступления, а в момент наименьшей загрузки. Это может быть определенное время суток (например, ночь) или действительно момент небольшой загрузки, определяемый сервером репликации. Конечно, такой подход увеличивает время, в течение которого данные на серверах будут рассогласованы, так что этот вопрос остается «на совести» администратора системы или проектировщика приложения.
- Сложность администрирования является стимулом для написания программы-администратора, которая может взять на себя ряд основных и наиболее часто требующихся функций: чистка хранилища (в зависимости от накопившихся данных или по времени), отслеживание коллизий, проверка и синхронизация реплицируемых таблиц и многое другое.
- Возможно усложнение правил репликации, введение горизонтальной (только выборочные записи) и вертикальной (выборочные столбцы) репликации.
- Рассылка произведенных изменений может быть синхронной (с ожиданием ответа) и асинхронной – изменения рассылаются, а ответ когда придет – тогда и придет. Второй способ быстрее, но не всегда гарантирует последовательное выполнение транзакций (хотя принципиально этот вопрос решаем).

Итак, после анализа достоинств и недостатков асинхронной репликации можно сделать вывод, что в не очень динамичных приложениях система асинхронной репликации является практически оптимальным решением, которое не предъявляет слишком больших требований к аппаратуре и, следовательно, выигрывает и с точки зрения соотношения цена/производительность. Наиболее целесообразно применять асинхронную репликацию в приложениях, которые предъявляют высокие требования к производительности поисковых запросов и не критичны к временному расхождению данных.

XIII. Система синхронизации DBSync

DBSync – это программное решение, предназначенное для построения распределенных прикладных систем, работающих с реляционными базами данных. Оно обеспечивает возможность off-line работы с информацией для удаленных и мобильных клиентов, а также последующей двунаправленной синхронизации данных с центральным сервером. Связь с базой данных осуществляется через ODBC, поэтому на базе DBSync можно строить решения для гетерогенных систем.

DBSync содержит следующие основные компоненты: механизм определения изменений, модуль формирования пакетов синхронизации, модуль обработки пакетов синхронизации, транспортный уровень и модуль интеграции со службой каталогов. DBSync может быть запущен в режиме клиента (та сторона, которая инициирует процесс синхронизации баз данных) и в режиме сервера (та сторона, которая ожидает начала процесса синхронизации данных).

Структура DBSync показана на рисунке 2.

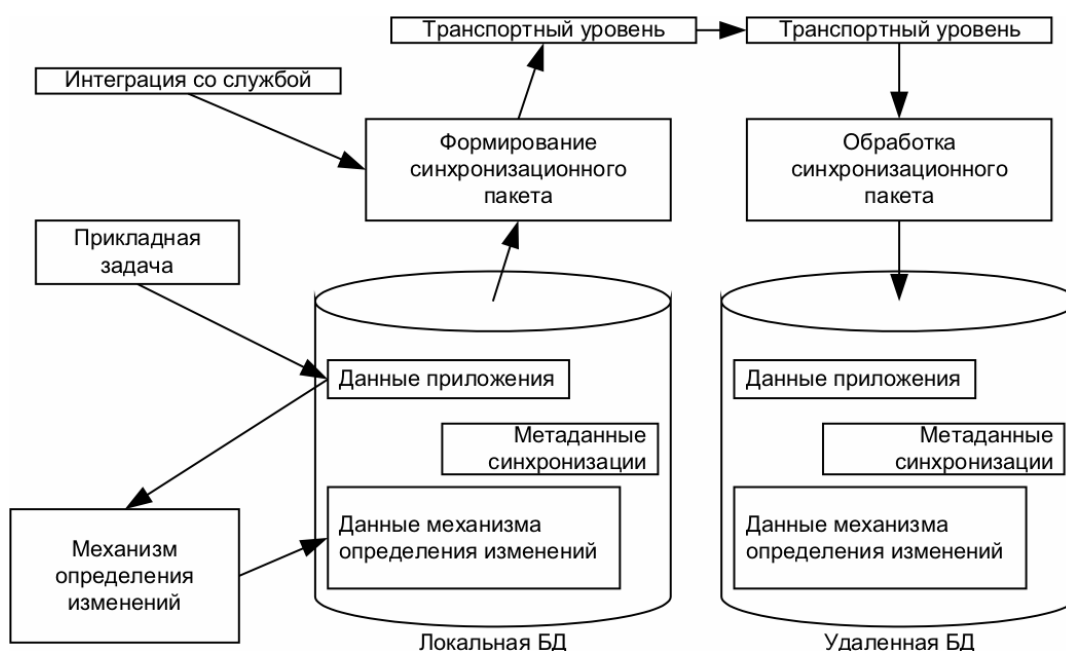


Рисунок 2 Структура DBSync

При установке DBSync в базе данных создается набор таблиц с метаданными синхронизации. В метаданных хранятся списки узлов синхронизации (удаленных баз данных), списки таблиц, которые необходимо синхронизировать, правила синхронизации и т.д. Кроме того, для прикладных таблиц, которые включены в процесс синхронизации данных, создаются вспомогательные таблицы.

Вспомогательные таблицы используются механизмом определения изменений для хранения служебной информации. Механизм определения изменений базируется на триггерах. Когда прикладная задача модифицирует данные, информация об этом событии заносится во вспомогательные таблицы. При этом объем служебной информации ограничен и не увеличивается при многократных изменениях одной и той же записи. Синхронизация локальной базы с удаленным узлом возможна в режимах on-line и off-line. В режиме on-line клиент синхронизации устанавливает

соединение с сервером синхронизации по одному из поддерживаемых on-line протоколов (TCP/IP, HTTP и т.д.). Затем формируется пакет синхронизации, передается на удаленный сервер БД и там обрабатывается. В зависимости от обстоятельств пакет синхронизации может содержать либо все данные (начальная синхронизация, восстановление после сбоя), либо данные, которые были модифицированы с момента последнего сеанса синхронизации с данным узлом. Модуль интеграции со службой каталогов определяет, какие именно таблицы и по каким условиям будут выгружены в удаленную базу. После успешной обработки пакета удаленной стороной в локальную базу возвращаются изменения, которые произошли в удаленной базе. На этом сеанс синхронизации завершается. В случае off-line протокола алгоритм обмена пакетами зависит от настроек DBSync.

DBSync обеспечивает возможности обмена данными между синхронизируемыми базами данных при помощи следующих сетевых протоколов:

TCP/IP – наиболее быстрое и надежное решение. Обеспечивает дуплексную сессию синхронизации. При необходимости шифрации сообщений имеется возможность работать через SSL.

HTTP – аналогично TCP/IP обеспечивает получение подтверждений о приходе пакетов от базы-приемника, и сохранения пакетов не требуется. Серверная часть (принимающая соединения) реализована в варианте собственного сервера, понимающего подмножество http. Главным преимуществом при использовании этого протокола является возможность работы через Internet.

E-MAIL – Данный вариант обмена не гарантирует надежной доставки данных, но позволяет использовать любые возможности для отправки данных, т.к. e-mail является очень распространенным сервисом.

FILE – Имеет такие же ограничения, как и E-MAIL. Пакет синхронизации данных оформляется в виде файла. Доставка файла может осуществляться по ftp, с помощью гибких носителей и т.п.

В качестве прочих достоинств DBSync следует отметить:

- работу с UNICODE;
- формат сообщений XML;
- простоту администрирования;
- гибкость в конфигурировании.

XIV. Будущее системы ЛИНТЕР®

Будущее системы ЛИНТЕР связывается с новой версией СУБД ЛИНТЕР – ЛИНТЕР v.6.2. Эта версия обещает быть более мощной во всех отношениях.

Многое из того, что войдёт в эту систему уже реализовано и проходит тестирование.

А. Мобильность

Планируется перенос ЛИНТЕР на МОС-ЕС (S/390).

В. Масштабируемость

Здесь основным направлением является внедрение многопроцессорного ядра СУБД ЛИНТЕР 6.2. Подобное ядро использует порождение процессов (нитей) обработки запроса или его каких-то его фрагментов.

Таким образом, решается проблема распараллеливания обработки не только запросов, выполняющихся по различным каналам, но даже и обработки одного запроса.

Вторым направлением масштабируемости является эффективное использование больших объемов оперативной памяти (более 2 Gb).

С. Конфиденциальность информации

Центральным звеном здесь станет шифрация информации в таблице и её индексах.

Кроме того, в качестве дополнительной защиты рассматривается возможность реализации дополнительной аутентификации пользователя при помощи электронного ключа.

Д. Производительность (оптимизация, индексы)

До настоящего момента основными направлениями развития СУБД ЛИНТЕР были надёжность и секретность. Теперь настало время для увеличения быстродействия.

В этом направлении планируется включить в СУБД ЛИНТЕР возможность индексация многомерных пространственных объектов (R⁺, X-, M- деревья) для ускорения работы с пространственными данными типа карт, выкроек и пр..

Е. Сетевые компоненты

Выше уже упоминалось о том, что сетевые компоненты системы отслеживают «живучесть» тех объектов, с которыми они связаны. «Смерть» этих объектов будет вовремя обнаружена, и приложению (или ядру ЛИНТЕР) будет сообщено о невозможности продолжения работы.

Однако во многих случаях это не лучший выход. Особенно когда речь идёт о высоконадёжных системах, где необходимо неоднократно убедиться в том, что связь с объектом невозможна, несколько раз повторить действие (причём через установленные промежутки времени) и только потом передавать проблему для решения наверх.

F. Архиватор – lhb

В lhb добавятся функции сохранения данных в псевдотекстовом формате и формирование блоков базы данных при восстановлении (быстрая загрузка) для обеспечения переноса между версиями.

G. Реальное время

В ближайшее время добавится ещё одно средство реального времени – *тайм-ауты блокировок*. Допустимым временем ожидания блокировки можно будет управлять из приложения пользователя.

Ещё один планируемый тайм-аут – это тайм-аут ожидания выполнения запроса. Запрос, не обработанный за указанное время, будет автоматически снят с выполнения, и приложение получит управление с соответствующим кодом завершения.

H. Полнотекстовая индексация

Планируется расширить возможности полнотекстовой индексации путем реализации индекса по NCHAR полям и добавления возможности определения позиции найденного слова или фразы в исходном документе.

XV. Информация о разработчике

Компания РЕЛЭКС – один из ведущих российских разработчиков системного и прикладного программного обеспечения.

Основанная в 1990 году, компания стала первым в России независимым разработчиком систем управления базами данных.

РЕЛЭКС предлагает полнофункциональные решения на базе собственных сертифицированных средств разработки, поддержки и эксплуатации автоматизированных систем различного уровня сложности: от простых приложений до сложнейших защищенных информационных систем общегосударственного масштаба, средств накопления и анализа информации, встроенных, мобильных систем, систем синхронизации данных и систем реального времени.

Решения, разработанные специалистами РЕЛЭКС, на протяжении многих лет используются государственными структурами России, российскими и зарубежными коммерческими компаниями. Среди наших постоянных клиентов компании России, США, Израиля и Японии.

A. Товарные знаки

РЕЛЭКС, ЛИНТЕР®, НЕВОД®, DBSync являются товарными знаками, принадлежащими ЗАО НПП «Реляционные экспертные системы» (далее по тексту – компания РЕЛЭКС). Прочие названия и обозначения продуктов являются товарными знаками их производителей, продавцов или разработчиков.

B. Интеллектуальная собственность

Правообладателем продуктов ЛИНТЕР®, НЕВОД®, DBSync является компания РЕЛЭКС (1990–2006). Все права защищены. Данный документ является собственностью компании РЕЛЭКС. Ни одна часть данного документа не может быть воспроизведена, передана, преобразована, сохранена в системе поиска информации, переведена на другой язык или компьютерный язык в какой-либо форме, какими-либо средствами, электронными, механическими, магнитными, оптическими, химическими, ручными или иными без предварительного разрешения компании РЕЛЭКС.

C. О документе

Материал, содержащийся в данном документе, прошел тщательную проверку, но компания РЕЛЭКС не гарантирует, что документ не содержит ошибок и пропусков. Компания РЕЛЭКС оставляет за собой право в любое время вносить в документ исправления и изменения, пересматривать и обновлять содержащуюся в нем информацию.

D. Контактная информация**ЗАО НПП «РЕЛЭКС»**

| | |
|----------|--|
| Адрес: | Россия, 394006, г. Воронеж, ул. 20-летия Октября, 119 |
| Телефон: | (4732) 711-711; 778-333 |
| E-mail: | market@relex.ru linter@relex.ru |
| Http: | http://www.relex.ru |

E. Техническая поддержка

Отдел поддержки и сопровождения программных продуктов:

| | |
|----------|--|
| Телефон: | : (4732) 711–711 с 9:00 до 18:00 |
| E-mail: | support@relex.ru |
| | market@relex.ru |

С целью повышения качества программных продуктов и предоставляемых услуг в компании РЕЛЭКС действует система обработки рекламаций. Обо всех обнаруженных недостатках и/или ошибках в программном обеспечении или документации компании РЕЛЭКС просим сообщить, посетив Интернет-страницу <http://bt.relex.ru> (наша система обработки рекламаций).