Базы данных (часть 2)

Кореньков Владимир Васильевич
профессор САУ,
зав. кафедры «Распределенные информационно-вычислительные системы»,
зам. директора Лаборатории информационных технологий ОИЯИ

Основные темы лекций по курсу СУБД

1. Основные понятия баз данных. Этапы развития СУБД. Требования к системам управления базами данных.
2. Архитектура баз данных. Логическая и физическая независимость данных. Схема прохождения запросов к БД. Режимы работы с базой данных. Схема прохождения запроса к БД. Классификация моделей данных. Архитектура и модели "клиент-сервер" в технологии БД.
3. Реляционная модель БД. Таблица, кортеж, атрибут, домен, первичный ключ, внешний ключ. Основные достоинства реляционной модели. Фундаментальные свойства отношений. Обеспечение целостности данных.
4. Операторы реляционной алгебры. Понятия полной и транзитивной функциональной зависимости. Нормализация, нормальные формы.
5. Проектирование баз данных. Семантические модели данных. ER - модель (Entity-Relationship, Сущность-Связи). Этапы проектирования баз данных.
6. Язык SQL, его структура, стандарты, история развития. Подмножество языка DML: операторы SELECT, INSERT, UPDATE, DELETE.
7. Подмножество языка DDL: операторы CREATE, ALTER, DROP. Поддержка ссылочной целостности данных. Представления, их значение. Обновляемые представления.
8. Объектные и системные привилегии. Операторы GRANT, REVOKE. Роли. Транзакции. Операторы управления транзакциями: COMMIT, ROLLBACK, SAVEPOINT. Журнал транзакций.

Темы лекций

- 1. Язык PL/SQL, его структура, основные операторы.
- 2. Курсоры, операторы работы с курсором, оператор SELECT INTO.
- 3. Процедуры, функции, пакеты.
- 4. Триггеры, их основные свойства и значение.
- 5. Параллельные архитектуры БД; масштабируемость, надежность, производительность.
- 6. Распределенные базы данных, фрагментация, тиражирование.
- 7. Средства защиты данных в СУБД.
- 8. Шлюзы к базам данных. Архитектура ODBC. WWW-интерфейс к БД.
- 9. Объектная модель данных
- 10. Объектно-ориентированные и объектно-реляционные БД.
- 11. СУБД ORACLE (технологии и возможности ORACLE 8i, ORACLE 9i, ORACLE 10g)
- 12. GRID-технологии. Перспективы развития технологий баз данных.

Литература

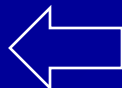
- Дейт К. Введение в системы баз данных. – 8 издание, М., Вильямс, 2005
- Т. Конноли, К. Бегг, А. Страхан «Базы данных. Проектирование, реализация и сопровождение. Теория и практика». – М: «Вильямс», 2000
- Т. Карпова Базы данных. Модели, разработка, реализация. – СПб: Питер, 2001
- Бобровски С. Oracle 7 и вычисления клиент/сервер. – М.: Лори, 1995
- Бобровски С. Oracle 8: Архитектура – М.: Лори, 1998
- И. Баженова Oracle 8/8i. – М., Диалог-МИФИ,2000
- Генник Д. Справочник по SQL. – М.: Питер, 2004
- Фейерштейн С., Прибыл Б., Доз Ч. Oracle PL/SQL – М:, Питер, 2004
- А. Саймон Стратегические технологии баз данных. – М., Финансы и статистика, 2000
- М.Р. Когаловский «Энциклопедия технологий баз данных». – М:, Финансы и статистика, 2002
- Синлор Р., Стегман М. Использование ODBC для доступа к базам данных. – М.: BINOM, 1995
- www.osp.ru www.osp.ru, www.citforum.ru

PL/SQL (Procedural Language) — процедурное расширение языка SQL

PL/SQL - Procedural Language. Как видно из его названия, PL/SQL расширяет возможности SQL, добавляя в него конструкции процедурных языков, такие как:

- ✔ Переменные и типы (как предварительно определенные, так и определяемые пользователем).
- ✔ Управляющие структуры, такие как операторы и циклы IF-THEN-ELSE.
- ✔ Процедуры и функции.
- ✔ Объектные типы и методы (PL/SQL версии 8 и выше).

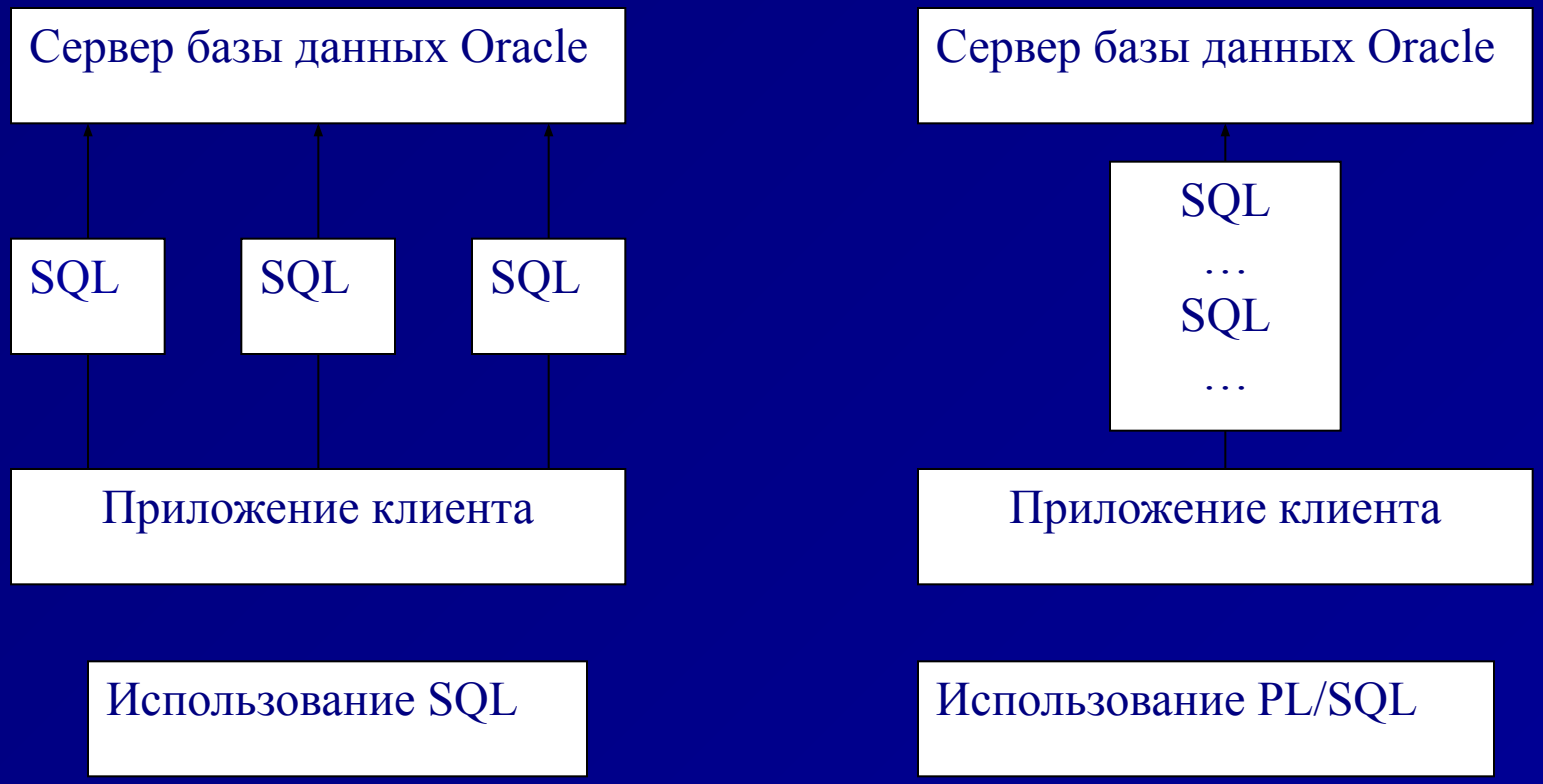
PL/SQL — полноценный язык программирования, не являющийся самостоятельным продуктом. Это технология, включающая механизм, выполняющий блоки PL/SQL (PL/SQL engine). Механизм может быть встроен в ядро СУБД, или же в любое инструментальное средство Oracle.



PL/SQL уникален тем, что соединяет гибкость SQL с мощью и способностью к конфигурированию языка 3GL. В нем имеются как необходимые процедурные конструкции, так и возможность обращения к базе данных. Таким образом, этот язык программирования является надежным, эффективным языком, хорошо подходящим для разработки сложных приложений.

Основное отличие PL/SQL от других процедурных языков заключается во встроенном механизме обработки курсоров, позволяющем выполнять операторы непроцедурного языка запросов SQL из PL/SQL-программы.

Модель клиент — сервер



PL/SQL в среде клиент /сервер

Многие приложения для работы с базами данных создаются с использованием модели клиент /сервер.
Сама программа размещается на компьютере клиента и посылает запросы на получение информации серверу базы данных. Запросы инициируются при помощи SQL, что приводит к наличию в сети большого числа посылок — по одной на каждый SQL-оператор.

Несколько SQL-операторов могут быть объединены в единый блок PL/SQL и посланы серверу как единое целое. В результате сетевой трафик снижается, а приложение функционирует намного быстрее.

Блок PL/SQL

Базовой единицей PL/SQL является блок (block). Все программы PL/SQL состоят из блоков, которые могут быть вложены один в другой.

Блок имеет следующую структуру:

DECLARE

<Раздел объявлений переменных, типов, курсоров и логических подпрограмм PL/SQL >

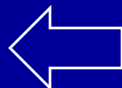
BEGIN

<Выполняемый раздел – процедурные и SQL- операторы. Это основной раздел блока и единственный, являющийся обязательным. >

EXCEPTION

<Раздел исключительных ситуаций – операторы обработки ошибок. >

END;



Допустимы следующие виды блоков:

- ***Анонимные (непоименованные) блоки*** создаются, как правило, динамически и выполняются только один раз
- ***Именованные блоки*** – это анонимные блоки с метками, дающими блокам имена. Они также создаются как правило, динамически и выполняются только один раз.
- ***Подпрограммы*** – это процедуры, модули и функции, хранимые в базе данных. Эти блоки, как правило, не изменяются и выполняются многократно явным образом посредством вызова процедуры, модуля или функции.
- ***Триггеры*** – это именованные блоки, которые также хранятся в базе данных. Они тоже, как правило, не изменяются и выполняются многократно неявным образом при наступлении соответствующих событий. Событием, вызывающим активизацию триггера, является оператор языка DML, выполняемый над некоторой таблицей базы данных.

Замечание *При создании процедуры ключевое слово DECLARE необязательно. Более того, его использование будет ошибкой. Однако DECLARE требуется при создании триггера.*

Лексические единицы

Набор символов PL/SQL

При работе с PL/SQL допускается использование символов из определенного набора знаков. В этот набор входят почти все символы, которые можно ввести с клавиатуры. Однако существуют ограничения на применение ряда символов в некоторых конкретных ситуациях.

Набор символов, который можно использовать при программировании на PL/SQL:

- Все прописные и строчные буквы
- Цифры от 0 до 9
- Знаки ( ) + √ * / > < = ! ~ ; : . ' @ % , " # $ ^ & _ { } ? [ ]

1. Арифметические операторы

Оператор	Операция	Оператор	Операция
+	сложение	/	деление
-	вычитание	**	возведение в степень
*	умножение		

2. Операторы сравнения

Оператор	Операция	Оператор	Операция
<>	не равно	<	меньше
!=	не равно	>	больше
^=	не равно	=	равно

Идентификаторы

Идентификаторы используются для именования переменных, курсоров, типов и подпрограмм. При выборе идентификаторов следует руководствоваться следующими правилами:

- **Идентификатор должен начинаться с буквы (A-Z).**
- **За первой буквой переменной может следовать одна или несколько букв, цифр (0-9) или специальных символов $, # или _ .**
- **Длина идентификатора не может превышать 30 символов.**
- **Идентификатор не может содержать пробелы.**

Пример:

```
currentcustomer CHAR(15);      -- переменная
```

Константы

При объявлении константы указывается CONSTANT , а после идентификатора типа – оператор присваивания и значение константы.

```
discont CONSTANT REAL := 0.1;      -- константа
```

Типы данных

Тип	Подтип
NUMBER (precision, scale)	DECIMAL, REAL, FLOAT, NUMERIC INTEGER, SMALLINT,
CHAR (length)	
VARCHAR2 (length)	
DATE	
BOOLEAN	
RECORD	- составной тип данных
TABLE	- составной тип данных

Пример:

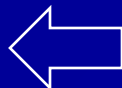
```
DECLARE
    TYPE  orderrecordtype IS RECORD
          ( id number(5,0) NOT NULL :=0,
            customerid NUMBER(5,0) NOT NULL :=0,
            orderdate DATE NOT NULL :=SYSDATE);
```

Записи PL/SQL

Записи (records) PL/SQL аналогичны структурам языка C. С помощью записи можно работать с несколькими отдельными, но связанными переменными как с одной программной единицей.

Объявим тип записи для хранения информации о студентах.

```
DECLARE
TYPE t_StudentRecord1 IS  RECORD (
        StudentID NUMBER(5);
        FirstName  VARCHAR2(20);
        LastName  VARCHAR2(20));
TYPE t_StudentRecord2 IS  RECORD (
        StudentID NUMBER(5);
        FirstName  VARCHAR2(20);
        LastName  VARCHAR2(20));
/* объявим переменные с этими типами*/
vStudentInfo1   t_StudentRecord1 ;
vStudentInfo2   t_StudentRecord2 ;
```



Чтобы присвоить одной записи значение другой они должны быть одного типа.

Хотя записи имеют одинаковые имена и типы полей, типы собственно записей различны, поэтому такая операция присваивания t_StudentRecord1 := t_StudentRecord2 неверна.

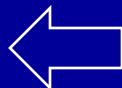
Однако типы полей совпадают, поэтому следующие операции верны:

```
t_StudentRecord1. StudentID := t_StudentRecord2 . StudentID;
t_StudentRecord1. FirstName := t_StudentRecord2 . FirstName;
```

Управляющие структуры PL/SQL

Структуры управления являются основой любого языка программирования, поскольку большинство реальных приложений должно уметь обрабатывать множество различных ситуаций. Основную часть структур управления выполнением программы составляют различного рода условные операторы, способные обнаружить существование той или иной ситуации, а за тем инициировать выполнение необходимых действий.

Управление ходом выполнения программы. Конкретная последовательность выполнения различных операторов программы определяется значениями ее переменных и содержанием информации, читаемой из базы данных и записываемой в нее.



Три типа условного оператора IF. При написании компьютерных программ неоднократно возникают ситуации, когда требуется проверить выполнение того или иного условия и в случае, если оно выполняется (имеет место логическое значение TRUE), осуществить одни действия, а при невыполнении условия (логическое значение FALSE) - другие. В языке PL/SQL предусмотрено три типа условного оператора if:

- **Конструкция IF-THEN**. Эта форма условного оператора предназначена для проверки простых условий. Если условие верно (TRUE), то выполняется одна или несколько строк программы, указанных в теле оператора. Если условие не выполняется (FALSE), то управление передается на следующий оператор.
- **Конструкция IF-THEN-ELSE.** Эта форма условного оператора аналогична предыдущей,но при невыполнении условия (FALSE) управление передается на один или несколько операторов, указанных после ELSE.
- **Конструкция IF-THEN-ELSIF.** Этот формат является альтернативой использованию вложенных операторов IF-THEN-ELSE.

Пример

```
DECLARE
        V_num1   NUMBER;
        V_num2   NUMBER;
        V_REZ    VARCHAR2(7);
BEGIN
        …..
          IF    V   _num1 < V_num2
          THEN
                  V_REZ := ‘YES’;
          ELSE
                  V_REZ := ‘NO’;
          END   IF;
END;
```

Пример

```
IF quantity > 15
      THEN …;     -- скидка 15%
   ELSIF quantity > 10
      THEN …;    -- скидка 10%
   ELSIF quantity > 5
      THEN …;    -- скидка 5%
   ELSE  …;       -- нет скидки
  ENDIF
```

Циклы

Четыре вида операторов цикла. Циклы позволяют организовать многократное выполнение одного и того же участка программы до полного завершения обработки.

Конструкция LOOP-EXIT-END LOOP

Пример:

```
DECLARE
      V_Counter  INTEGER := 1;
BEGIN
      LOOP
      INSERT INTO   temp_table  VALUES
      (V_Counter, 'LOOP index');
      V_Counter := V_Counter  + 1;
            IF    V_Counter  > 50   THEN
                  EXIT;
                  END   IF;
      END    LOOP;
END;
```

Конструкция LOOP-EXIT WHEN-END LOOP

Оператор EXIT WHEN условие эквивалентен оператору : IF условие THEN EXIT; END IF;

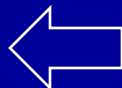
Пример:

```
DECLARE
    V_Counter  INTEGER := 1;
BEGIN
    LOOP
     INSERT INTO   temp_table  VALUES
     (V_Counter, 'LOOP index');
     V_Counter := V_Counter  + 1;
    EXIT   WHEN V_Counter  > 50
    END    LOOP;
END;
```

Конструкция WHILE-LOOP-END LOOP

Пример:

```
DECLARE
        V_Counter  INTEGER
    BEGIN
      WHILE   V_Counter <= 50  LOOP
            INSERT INTO   temp_table  VALUES
       (V_Counter, 'LOOP index');
            V_Counter := V_Counter  + 1;
      END    LOOP;
    END;
```



Конструкция FOR-IN [REVERSE] -LOOP-END LOOP

Пример: BEGIN

FOR V_Counter IN 1..50 LOOP

INSERT INTO temp_table VALUES

(V_Counter, 'LOOP index');

END LOOP;

END;

При использовании REVERSE (обратный порядок) индекс цикла будет изменяться от верхней границы до нижней, в следующем примере цикл начнется с 50 и каждый раз будет уменьшаться на 1.

Пример: BEGIN

FOR V_Counter IN REVERSE 1..50 LOOP

INSERT INTO temp_table VALUES

(V_Counter, 'LOOP index');

END LOOP;

END;

Верхняя и нижняя границы цикла могут быть любыми выражениями, для которых возможно преобразование в числовые значения.

Присваивание переменным значений базы данных

В зависимости от числа возвращаемых запросом строк используются два метода.

SELECT … INTO … - когда возвращается 1 строка

```
BEGIN
    SELECT id, customerid, orderdate
        INTO currentorder.id, currentorder.customerid,
                currentorder.orderdate
        FROM orders
        WHERE id=453;
```

Если по запросу возвращаются несколько строк, нужно воспользоваться курсором.

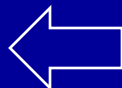
Курсоры

Курсор - это указатель на контекстную область с помощью которого программа PL/SQL может управлять контекстной областью и ее состоянием во время обработки оператора.

В языке PL/SQL курсоры используются для управления обработкой SQL-операторов select.

Курсоры представляют собой области памяти, специально предназначенные для обработки этих операторов.

В одних случаях курсоры объявляются явно, а других программист предоставляет PL/SQL самому выполнить эту операцию.



Явно объявляемые курсоры

Явное объявление курсора производится в секции DECLARE, причем указанный в определении SQL-оператор может содержать команды select.

Команды insert, update или delete здесь не допускаются. Явные курсоры используются для обработки тех операторов, которые возвращают более одной строки.

Обработка явных курсоров

Для обработки явного курсора в PL/SQL необходимо выполнить 4 шага:

1. Объявить курсор.
2. Открыть курсор для запроса.
3. Выбрать результаты в переменные PL/SQL.
4. Закрыть курсор.

Обработка явных курсоров

1) Объявление курсора

При объявлении курсора ему назначается имя и ставится в соответствие некоторый оператор SELECT.

Синтаксис объявления курсора таков:

CURSOR *имя_курсора* IS *оператор_select*

где *имя_курсора* - это имя курсора, *оператор_select* - запрос, который будет обрабатываться.

2) Открытие курсора для запроса

Синтаксис открытия курсора таков:

OPEN *имя_курсора;*

где *имя_курсора* - предварительно объявленный курсор.

Когда курсор открывается, происходит следующее:

- ✔ Анализируются значения переменных привязки.
- ✔ На основе значений переменных привязки определяется активный набор.
- ✔ Указатель активного набора устанавливается на первую строку.

3) Выбор результатов в переменные PL/SQL

Производится считывание строк из курсора. Частью оператора FETCH является список INTO. Оператор FETCH имеет две формы:

FETCH *имя_курсора* INTO *список_переменных*;

или

FETCH *имя_курсора* INTO *запись_ PL/SQL;*

где *имя_курсора* - обозначает предварительно объявленный и открытый курсор,

список_переменных - представляет собой список предварительно объявленных переменных PL/SQL, разделенных запятыми,

запись_ PL/SQL - предварительно объявленная запись PL/SQL.

Переменные в конструкции INTO должны иметь тип, совместимый со списком выбора запроса.

4) Закрытие курсора

Когда выбран весь активный набор, курсор следует закрыть.

Это означает, что программа закончила работу с курсором и отведенные для него ресурсы могут быть освобождены.

Синтаксис закрытия курсора таков:

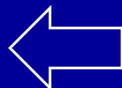
CLOSE *имя_курсора;*

где *имя_курсора* - ранее открытый курсор.

Курсорные атрибуты

В PL/SQL существует 4 атрибута, которые применимы к курсорам:

- ✔ **%FOUND** – это логический атрибут. Он возвращает TRUE, если при предшествующем считывании была выбрана строка, FALSE – если строка выбрана не была.
- ✔ **%NOTFOUND** ведет себя противоположно %FOUND. Этот атрибут часто используется в качестве условия выхода из цикла выборки.
- ✔ **%ISOPEN** – этот логический атрибут используется для определения, открыт или нет соответствующий курсор. Если открыт, то возвращает TRUE.
- ✔ **%ROWCOUNT** – этот числовой атрибут возвращает число строк, считанных курсором на данный момент.



Неявно объявляемые курсоры

Оператор select указывается в теле блока, и PL/SQL берет на себя всю заботу об определении курсора, выполняя соответствующие действия неявно. При этом программисту не требуется вносить в секцию DECLARE никаких дополнительных объявлений.

Обработка неявных курсоров

Каждый оператор select выполняется в пределах контекстной области и поэтому имеет курсор, указывающий на конкретную контекстную область. Такой курсор называется SQL-курсором. В отличии от явных курсоров SQL-курсор не открывается и не закрывается программой. PL/SQL неявно открывает SQL-курсор, обрабатывает SQL- оператор и в последствии закрывает этот курсор, поэтому команды OPEN, FETCH, CLOSE не нужны.

Неявные курсоры используются для обработки операторов INSERT, UPDATE, DELETE, а также однострочных операторов SELECT…INTO

Пример явного(explicit) курсора

```
DECLARE
        /*Выходные переменные для хранения результатов запроса */
              v_StudentID  students. Id%TYPE;
              v_FirstName students. fitst_name%TYPE;
              v_ LastName students. last_name%TYPE;
        /*  Переменная привязки, используемая в запросе*/
            v_Major    students.major%TYPE := 'Computer Science';
        /*  Создание курсора*/
        CURSOR c_Students IS
              SELECT  id, first_name, last_name
              FROM students
                    WHERE   major =  v_Major;
BEGIN
         /*Обозначим строки активного набора*/
              OPEN c_Students
             LOOP
         /*Выберем каждую строку  активного набора в переменные PL/SQL*/
                   FETCH c_Students  INTO v_StudentID, v_FirstName, v_ LastName;
         /*  Если строки, которые нужно выбрать, закончились, выйдем из цикла*/
                    EXIT  WHEN c_Students%NOTFOUND;
              END  LOOP;
        /*  Освободим ресурсы, используемые запросом*/
              CLOSE   c_Students ;
END;
```

Пример неявного(implicit) курсора

```
BEGIN
        UPDATE    rooms
                  SET   number_seats = 100
                   WHERE   room_id = 999;
/*   Если предыдущий оператор  UPDATE не выбирает ни одной строки, то введем новую
     строку в таблицу rooms*/
        IF     SQL%NOTFOUND THEN
               INSERT      INTO   rooms (room_id, number_seats)
               VALUES (999,  100);
        END IF;
END;
```

Эту же задачу можно выполнить при помощи атрибута SQL%ROWCOUNT:

```
BEGIN
        UPDATE    rooms
                  SET   number_seats = 100
                   WHERE   room_id = 999;
/*   Если предыдущий оператор  UPDATE не выбирает ни одной строки, то введем новую
     строку в таблицу rooms*/
              IF     SQL%ROWCOUNT     THEN
               INSERT      INTO   rooms (room_id, number_seats)
               VALUES (999,  100);
             END IF;
END;
```

Примеры

CURSOR *ordercursor* IS *select id, customerid, orderdate from orders;*

DECLARE
CURSOR *ordercursor (ordernumber NUMBER)* IS
SELECT id, customerid, orderdate FROM orders
WHERE id > ordernumber;
BEGIN
OPEN *ordercursor (3)*

В данном примере возвращаемый набор *ordercursor* включает строки таблицы *orders*, для которых идентификатор id > 3

Оператор GOTO

GOTO <метка> - оператор безусловного перехода

Обработка ошибок (блок EXCEPTION)

PL/SQL имеет встроенные исключительные ситуации

no_data_found, too_many_rows, invalid_number, …

EXCEPTION

  When *no_data_found* **then**

    …

  When *too_many_rows* **then**

    …

  END

Процедура **RAISE_APPLICATION_ERROR (ERRNUM, ERRMES)**

ERRNUM – пользователь задает номер ошибки от -20000 до -20999

Процедуры

Создание процедуры

Синтаксис оператора CREATE OR REPLACE PROCEDURE таков:

CREATE [OR REPLACE] PROCEDURE *имя_процедуры*

[(*аргумент* [{IN | OUT |IN OUT}] *тип*,

. . .

***аргумент* [{IN | OUT |IN OUT}] *тип*}] {IS | AS}**

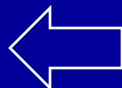
тело_процедуры

где ***имя_процедуры*** - это имя создаваемой процедуры,

аргумент - имя параметра процедуры,

тип - это тип соответствующего параметра,

тело_ процедуры - блок PL/SQL, в котором содержится текст процедуры.



Тело процедуры

Тело (body) процедуры - это блок PL/SQL, содержащий раздел объявлений, выполняемый раздел и раздел исключительных ситуаций. В описании процедуры ключевое слово DECLARE отсутствует.

Как и в анонимных блоках обязательным является только выполняемый раздел. Таким образом, структура процедуры такова:

```
CREATE OR REPLACE PROCEDURE имя_процедуры AS
                    /* Раздел объявлений. */
BEGIN
    /* Выполняемый раздел. */
EXCEPTION
    /* Раздел исключительных ситуаций. */
END [имя_процедуры];
```

Для изменения текста процедуры необходимо удалить и повторно создать ее. Во время разработки процедур эта операция выполняется достаточно часто, поэтому ключевые слова OR REPLACE (или заменить) позволяют выполнить такую операцию за один раз.

Ограничения на формальные параметры

При вызове процедуры ей передаются значения фактических параметров, и внутри процедуры к этим значениям обращаются с помощью формальных параметров. При этом передаются не только значения, но и ограничения, наложенные на переменные. Описывая процедуры, запрещается ограничивать длину параметров типа CHAR и VARCHAR2, а также точность и/или масштаб параметров типа NUMBER.

%TYPE и параметры процедур

Единственным способом наложения ограничения на формальные параметры является использование атрибута %TYPE. Если формальный параметр объявлен при помощи %TYPE, а базовый тип ограничен, это ограничение распространяется не на фактический параметр, а на формальный.

Значения параметров по умолчанию

Как и переменные, формальные параметры процедуры или функции могут иметь значения по умолчанию. В таком случае параметр можно не передавать из вызывающей среды. Если же параметр передается, вместо значения по умолчанию берется фактический параметр. Значение по умолчанию для параметра указывается следующим образом:

имя_параметра **[*вид*]** ***тип_параметра***

{:= | DEFAULT} ***исходное _значение***

где *имя _параметра* - это имя формального параметра, *вид* - вид параметра ((IN, OUT или IN OUT), *тип _параметра* - тип параметра, *исходное _значение* - значение, присваиваемое формальному параметру по умолчанию.

Можно применять или символы : = , или ключевое слово DEFAULT.

Удаление процедур

Процедуры и функции, как и таблицы, могут быть удалены. Синтаксис удаления процедуры выглядит следующим образом:

DROP PROCEDURE *имя _процедуры;*

Хранимые процедуры

Хранимые процедуры - приложение, объединяющее запросы и процедурную логику и хранящееся в базе данных. Хранимые процедуры позволяют содержать вместе с БД достаточно сложные программы, выполняющие большой объем работы без передачи данных по сети и взаимодействия с клиентом.

Пример процедуры

```
CREATE PROCEDURE deletecustomer (custid IN INTEGER) AS
    last VARCHAR2(50);
    first VARCHAR2(50);
BEGIN
   SELECT lastname, firstname INTO last, first
        FROM customer WHERE id=custid;
   INSERT INTO customerhistory VALUES (custid, last, first)
    DELETE FROM customer WHERE id=custid;
EXCEPTION
     WHEN no_data_found THEN
RAISE_APPLICATION_ERROR (-20123, ‘invalid Customer ID’)
END deletecustomer;
```

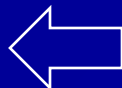
Функции

Создание функций

Функции очень похожи на процедуры. Как те, так и другие принимают аргументы, которые могут иметь любой вид.

Функции и процедуры - это различные формы блоков PL/SQL, в состав каждого из них могут входить раздел объявлений, выполняемый раздел и раздел исключительных ситуаций. Как функции, так и процедуры можно хранить в базе данных или описывать в блоке. Однако вызов процедуры сам по себе является оператором PL/SQL, в то время как вызов функции - это часть некоторого выражения.

Как и для процедур, список аргументов необязателен. В этом случае ни при описании функции, ни при ее вызове круглые скобки указывать не нужно. Однако тип, возвращаемый функцией, необходим, так как вызов функции является частью некоторого выражения. Тип функции используется для определения типа выражения, содержащего вызов этой функции.



Описание функций

Синтаксис для создания хранимой функции очень похож на синтаксис для создания процедуры:

CREATE [OR REPLACE] FUNCTION *имя_функции*

[(*аргумент* [{IN | OUT |IN OUT}] *тип*,

. . .

***аргумент*[{IN | OUT |IN OUT}] *тип*)]**

RETURN *возвращаемый _тип* {IS | AS} *тело_функции*

где *имя_функции* - это имя функции;

аргумент и *тип* аналогичны аргументу и типу, указываемым при создании процедуры;

возвращаемый_тип - это тип значения, возвращаемого функцией;

тело_функции - блок PL/SQL, содержащий программный текст данной функции.

Оператор RETURN

Внутри тела функции оператор RETURN применяется для возврата управления программой и результата выполнения функции в вызывающую среду. Общий синтаксис оператора RETURN выглядит следующим образом:

RETURN *выражение*,

где *выражение* - это возвращаемое значение. Значение выражения преобразуется к типу, указанному в команде RETURN при описании функции, если это значение уже не имеет данный тип. При выполнении оператора RETURN управление программой сразу же возвращается в вызывающую среду.

В функции может быть несколько операторов RETURN, хотя выполняться будет только один из них. Завершение функции без оператора RETURN является ошибкой.

Свойства функций

Многие из свойств функций аналогичны свойствам процедур:

- Функции могут возвращать более одного значения при помощи параметра вида OUT.
- Программный код функции состоит из раздела объявлений, выполняемого раздела и раздела исключительных ситуаций.
- Функции могут использовать значения по умолчанию.
- Функции можно вызывать, используя позиционное или именное представление.

Когда применять функцию, а когда процедуру зависит от того, сколько значений должна возвращать данная подпрограмма и как будут использоваться эти значения. Обычно принято следующее правило: если возвращается более одного значения, нужно использовать процедуру, а если ровно одно, то функцию.

Удаление функций

Процедуры и функции, как и таблицы, могут быть удалены. При выполнении этой операции процедура или функция удаляется из словаря данных. Синтаксис удаления функции выглядит следующим образом:

DROP FUNCTION *имя_функции*.

Пример функции

```
CREATE OR REPLACE FUNCTION AlmostFull(
    p_Department classes.department%TYPE,
    p_Course   classes.course%TYPE)
RETURN BOOLEAN IS
    V_CurrentStudents  NUMBER;
    V_MaxStudents   NUMBER;
    V_ReturnValue    BOOLEAN;
    V_FullPercent     CONSTANT   NUMBER := 90;
BEGIN       /*Узнаем текущее и максимальное число студентов в указанной группе*/
      SELECT  current_students, max_students
                INTO   V_CurrentStudents, V_MaxStudents
                FROM  classes
                WHERE  department = p_Department    AND   course = p_Course;
    /*Если процент заполнения группы более заданного  в V_FullPercent */

    IF   (V_CurrentsStudents / V_MaxStudents * 100) > V_FullPercent
    THEN
      V_ReturnValue := TRUE;
    ELSE
      V_ReturnValue := FALSE;
    END    IF;
    RETURN  V_ReturnValue;
END  AlmostFull;
```

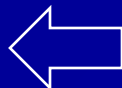
Пример функции

```
CREATE FUNCTION findcustid (last IN VARCHAR2, first IN VARCHAR2)
RETURN INTEGER  AS
  custid INTEGER;
BEGIN
    SELECT id INTO custid   FROM customer
      WHERE lastname=last AND firstname=first;
    RETURN custid;
EXCEPTION
     WHEN no_data_found THEN
          RAISE_APPLICATION_ERROR (-20101, ‘invalid Customer ID’)
END findcustid;
```

Агрегирующие функции

Групповые функции обрабатывают по несколько строк, но возвращают один результат. Эти функции можно применять только в списках выбора запросов и в конструкции GROUP BY.

В большинстве этих функций допускается использование квалификаторов (уточнителей) аргументов: DISTINCT (отличные от других) и ALL (все). Если указывается DISTINCT, рассматриваются только те значения, возвращаемые запросом, которые отличны от других. Когда используется квалификатор ALL, функция рассматривает все значения, возвращаемые запросом. Если не указано другое условие, ALL принимается как параметр, заданный по умолчанию.



MAX

Синтаксис	MAX([DISTINCT / ALL]
Назначение	Возвращает максимальное значение для пункта списка выбора
Область применения	Только списки выбора запросов и конструкции GROUP BY

COUNT

Синтаксис	COUNT([* / DISTINCT / ALL]
Назначение	Возвращает число строк в запросе. Если указана *, возвращается общее число строк. Если указан пункт сипска выбора, то подсчитываются не-NULL значения
Область применения	Только списки выбора запросов и конструкции GROUP BY

AVG

Синтаксис	AVG([DISTINCT / ALL]
Назначение	Возвращает среднее для значения столбца
Область применения	Только списки выбора запросов и конструкции GROUP BY

MIN

Синтаксис	MIN([DISTINCT / ALL]
Назначение	Возвращает минимальное значение для пункта списка выбора
Область применения	Только списки выбора запросов и конструкции GROUP BY

SUM

Синтаксис	SUM([DISTINCT / ALL]
Назначение	Возвращает сумму значений для пункта списка выбора
Область применения	Только списки выбора запросов и конструкции GROUP BY

Модули (Пакеты)

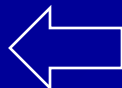
Модуль - это конструкция PL/SQL, позволяющая хранить связанные объекты в одном месте.

Модуль состоит из двух различных частей: описания и тела, каждая из которых хранится по отдельности в словаре данных.

В отличие от процедур и функций, которые содержатся локально в блоке или хранятся в базе данных, модули могут быть только хранимыми и никогда локальными.

Модули позволяют объединять связанные объекты, а также используют менее ограничений, определяемых зависимостями. Кроме того, они имеют ряд свойств, повышающих производительность системы.

В сущности, модуль представляет собой именованный раздел объявлений. Все входящее в состав раздела объявлений блока, может входить и в модуль: процедуры, функции, курсоры, типы и переменные. Размещение их в модуле позволяет ссылаться на них из других блоков PL/SQL, поэтому в модулях можно описывать глобальные переменные для PL/SQL.



Описание модуля

CREATE [OR REPLACE] PACKAGE *имя_модуля* {IS |AS}

описание_процедуры
описание_функции
объявление_переменной
определение_типа
объявление_исключительной_ситуации
объявление_курсора
END [*имя_модуля*];

где *имя_модуля* - это имя модуля. Элементы модуля (описания процедур и функции, переменные и т.д.) аналогичны указанным в разделе объявлении анонимного блока.

Для заголовка модуля верны те же синтаксические правила, установленные для раздела объявлений, за исключением объявлений процедуры и функции.

Перечислим эти правила:

1.Элементы модуля могут указываться в любом порядке. Однако, как и в разделе объявлений, объект должен быть объявлен до того, как на него будут произведены ссылки. Например, если частью условия WHERE курсора является некоторая переменная, то она должна быть объявлена до объявления курсора.

2.Присутствие элементов всех видов совсем не обязательно.

3.Объявления всех процедур и функций должны быть предварительными. В этом отличие модуля от раздела объявлений блока, где могут находиться как предварительные объявления, так и реальный текст процедур и функций.

Тело модуля

Тело модуля (package body) - это объект словаря данных, хранящийся отдельно от заголовка модуля. Тело модуля нельзя успешно скомпилировать без успешной компиляции заголовка.

В теле содержится текст подпрограмм, предварительно объявленных в заголовке модуля.

Тело модуля не является обязательной его частью. Если в заголовке не указаны какие-либо процедуры или функции (а только переменные, курсоры, типы и т.д.), тело можно не создавать.

Любое предварительное объявление в заголовке модуля должно быть раскрыто в его теле. Описание процедуры или функции должно быть таким же и включать в свой состав имя подпрограммы, имена ее параметров и вид каждого параметра.

Модули и области действия

Любой объект, объявленный в заголовке модуля, находится в области действия и видим вне границ этого модуля. Для обращения к объекту нужно указать имя модуля при ссылке на этот объект.
При этом вызов процедуры аналогичен вызову процедуры, не включенной в модуль. Единственное отличие такого вызова - присутствие перед именем процедуры имени модуля. Для модульных процедур могут задаваться параметры по умолчанию, и вызывать такие процедуры можно при помощи как позиционного, так и именного представления, то есть точно так же, как и обычные хранимые процедуры

Кроме того, в модуле можно применять типы данных, определяемые пользователями.

Инициализация модуля

При вызове первый раз модуль конкретизируется (instrantiated). Это значит, что модуль считывается с диска в память, а затем запускается p-код. В этот момент для всех переменных, описанных в модуле, выделяется память. У каждого сеанса будет собственная копия модульных переменных: это гарантирует, что два сеанса, выполняющие подпрограммы одного и того же модуля, будут использовать различные области памяти.

Во многих случаях код инициализации нужно запускать на выполнение при первой конкретизации модуля. Это можно сделать, если к телу модуля добавить раздел инициализации, разместив его после всех объектов:

CREATE OR REPLACE PACKAGE BODY *имя_модуля* {IS | AS}

. . .

BEGIN

код_инициализаци;

END [*имя_модуля*];

где *имя_модуля* – имя модуля,

***код_инициализации* – запускаемый код.**

Пример модуля генерации случайных чисел

```
CREATE OR REPLACE PACKAGE Random AS
PROCEDURE ChangeSeed (p_NewSeed IN NUMBER);
FUNCION Rand RETURN NUMBER;
PROCEDURE GetRand (p_RandomNumber OUT NUMBER);
FUNCION RandMax (p_MaxVal IN NUMBER) RETURN
   NUMBER;
PROCEDURE GetRandMax (p_RandomNumber OUT
   NUMBER, p_MaxVal IN NUMBER);
END Random;

CREATE OR REPLACE PACKAGE BODY Random AS
        v_Multiplier CONSTANT NUMBER := 22695477;
        v_Increment CONSTANT NUMBER := 1;
        v_Seed NUMBER :=1;
```

```
PROCEDURE ChangeSeed(p_NewSeed IN NUMBER) IS
BEGIN
    v_Seed := p_NewSeed;
END ChangeSeed;

FUNCTION Rand RETURN NUMBER IS /*Возвращает
    случайное число в диапазоне от 1 до 32767*/
BEGIN
    v_Seed :=MOD(v_Multiplier * v_Seed + v_Increment, (2 **
    32));
    RETURN BITAND (v_Seed/(2 ** 16), 32767);
END Rand;

PROCEDURE GetRand(p_RandomNumber OUT NUMBER)
    IS /*Аналогична функции Rand, но с процедурным
    интерфейсом*/
BEGIN
    p_RandomNumber := Rand;
END GetRand;
```

```
CREATE OR REPLACE PACKAGE ClassPackege AS
PROCEDURE AddStudent(p_StudentId IN Students. Id %TYPE,
p_Department IN classes.departmen%TYPE, p_Courses IN
    classes.course%TYPE );
PROCEDURE RemoveStudent(p_StudentId IN Students.ID%TYPE,
    p_Department IN classes.departmen%TYPE,
    p_Courses IN classes.course%TYPE );
E_studentNotRegistered EXCEPTION;

TYPE t_StudentIDTable IS TABLE OF Students.Id %TYPE
INDEX BY BINARY_INTEGER;

PROCEDURE ClassList(p_Department IN classes.departmen%TYPE,
p_Courses IN classes.course%TYPE , pIDS OUT t_StudentIDTable,
p_NumStudents IN OUT BINARY_INTEGER );
END ClassPackege;
```

В этом модуле содержится описание трех процедур, одного типа и исключительной ситуации.

```
FUNCTION RandMax(p_MaxVal IN NUMBER) RETURN NUMBER  IS
BEGIN
--Возвращает случайное целое число в диапазоне от 1 до p_MaxVal
    RETURN   MOD (Rand,  p_MaxVal)  +  1;
END  RandMax;

PROCEDURE  GetRandMax(p_RandomNumber   OUT   NUMBER,  p_MaxVal
    IN  NUMBER)  IS
BEGIN
    p_RandomNumber := RandMax (p_MaxVal);
END  GetRandMax;
BEGIN
/* Инициализация модуля.  Инициализируем  исходное  значение
    текущим временем  в секундах */
  ChangeSeed(TO_NUMBER (TO_CHAR(SYSDATE, 'SSSS')));
END  Random;
```

Для получения случайного числа можно просто вызвать Random.Rand. Последовательность случайных чисел зависит от исходного значения – для одного и того же исходного значения генерируются одинаковые последовательности.

Пример модуля (пакета)

```
CREATE OR REPLACE PACKAGE customermanager IS
PROCEDURE newcustomer (company IN VARCHAR2 DEFAULT null,
   last IN VARCHAR2, first IN VARCHAR2, …);
FUNCTION findcustid (last IN VARCHAR2, first IN VARCHAR2)
 RETURN INTEGER;
   PROCEDURE updatecustomer (custid IN INTEGER, fieldtype IN
   CHAR,   newvalue IN VARCHAR2);
PROCEDURE deletecustomer (custid IN INTEGER);
PROCEDURE deletecustomer (last IN VARCHAR2, first IN
   VARCHAR2);
END customermanager;

CREATE  OR REPLACE PACKAGE BODY customermanager AS
….
END customermanager;
```

Триггеры

Триггеры так же, как процедуры и функции, являются именованными блоками PL/SQL с разделом объявлений, выполняемым разделом и разделом исключительных ситуаций.

Подобно модулям, триггеры необходимо хранить в базе данных, а не локально в блоке.

Триггер выполняется неявно, всякий раз, когда происходит событие, запускающее этот триггер, причем использование аргументов не допускается.

Акт выполнения триггера называется его активизацией (firing).

Запускается триггер операцией DML (INSERT, UPDATE или DELETE), выполняемой над базой данных.

Создание триггеров

CREATE [OR REPLACE] TRIGGER *имя_триггера*
{BEFORE | AFTER} активизирующее событие **ON** *ссылка_на_таблицу*
[FOR EACH ROW [WHEN *условие_срабатывания*]]
тело_триггера;

где *имя_триггера* – имя триггера;

активизирующее событие – момент активации триггера;

ссылка_на_таблицу – таблица для которой создан триггер;

условие_срабатывания – если оно есть, то сначала оно вычисляется, и только если условие это истинно, срабатывает тело триггера;

тело_триггера – программный текст триггера;

BEFORE-триггеры используются для проверки правильности и/или модификации вводимых данных, например перевод в верхний регистр и тп.

AFTER-триггеры выполняют действия с данными уже обработанными командой DML, например протоколирование действий пользователя и др.

Триггеры можно использовать для:

- ✔ Реализации сложных ограничений целостности данных, которые невозможно осуществить через описательные ограничения, устанавливаемые при создании таблицы
- ✔ Слежения за информацией, хранимой в таблице, путем записи вносимых изменений и пользователей, вносящих эти изменения
- ✔ Автоматического оповещения других программ о том, что делать в случае изменения информации, содержащейся в таблице

Типы триггеров

- ❖ Тип триггера определяется тем, какое событие его активизирует: INSERT (ввод), UPDATE (обновление) или DELETE (удаление).
- ❖ Триггеры могут активизироваться до (BEFORE) или после (AFTER) операции, а также для строки или оператора.
- ❖ Триггеры могут активироваться для строки или оператора. Если триггер строковый, то он активизируется один раз для каждой из строк, на которые воздействует оператор, вызывающий срабатывание триггера. Если триггер операторный, то он активизируется один раз до или после оператора. Строковые триггеры содержат условие FOR EACH ROW (для каждой строки оператора) в описании триггера.

Элементы триггера

Обязательными элементами триггера являются его имя, активизирующее событие и тело. Условие WHEN необязательно.

Имена триггеров

Пространство имен триггеров (набор идентификаторов), разрешенных для использования в качестве имен объектов отличается от пространств имен других подпрограмм.

Для процедур, модулей и таблиц применяется одно и то же пространство имен, это значит, что в пределах одной схемы базы данных все объекты, использующие одно и то же пространство имен, должны иметь уникальные имена.

Например, модуль и процедура в одной схеме не могут иметь одинаковых имен, а триггер может иметь то же тмя, что и процедура или модуль. Однако в пределах одной схемы конкретное имя может быть дано только одному триггеру.

Имена триггеров – это идентификаторы базы данных, поэтому подчиняются стандартным правилам для идентификаторов.

Удаление и запрещение триггеров

Триггеры, как и процедуры, и модули, и функции, можно удалять. Синтаксис таков:

DROP TRIGGER *имя_триггера*;

Однако в отличие от процедур и функций, можно не удаляя триггер, запретить (disable) его использование. Когда триггер запрещен, он по-прежнему находится в словаре данных, но никогда не активизируется.

С помощью оператора **ALTER TRIGGER *имя_триггера* {DISABLE| ENABLE};** можно запретить или разрешить любой триггер.

Все триггеры таблицы выключаются/включаются командой:

ALTER TABLE имя_таблицы {DISABLE | ENABLE} ALL TRIGGERS;

Порядок активизации триггера

Триггер активизируется при выполнении оператора DML. Алгоритм выполнения DMLоператора таков:

1. Выполняется операторный триггер BEFORE (при его наличии).

2. Для каждой строки, на которую воздействует оператор:

a. Выполняется строковый триггер BEFORE (при его наличии);

b. Выполняется собственно оператор;

c. Выполняется строковый триггер AFTER (при его наличии);

3. Выполняется операторный триггер AFTER (при его наличии).

Триггер никак не проверяет данные, которые уже были в таблице до того момента, когда он был создан или включен.

Пример содержит все 4 вида триггеров UPDATE (BEFOR и AFTER, операторный и строковый) для таблицы classes.

Создадим числовую последовательность:

```
CREATE SEQUENCE  trigger_seq
  START  WITH   1
  INCREMENT  BY 1;
```

```
CREATE [OR REPLACE] TRIGGER classesBEstatement
   BEFORE UPDATE ON classes
BEGIN
   INSERT INTO temp_table (num_col, char_col)
          VALUES (trigger_seq.NEXTVAL, 'BEFORE
   ОПЕРАТОРНЫЙ ТРИГГЕР')
END classesBEstatement;

CREATE [OR REPLACE] TRIGGER classesAFstatement
   AFTER UPDATE ON classes
BEGIN
   INSERT INTO temp_table (num_col, char_col)
          VALUES (trigger_seq.NEXTVAL, 'AFTER
   ОПЕРАТОРНЫЙ ТРИГГЕР')
END classesAFstatement;
```

```
CREATE [OR REPLACE] TRIGGER classesBERow
    BEFORE UPDATE ON classes FOR EACH ROW
BEGIN
    INSERT INTO temp_table (num_col, char_col) VALUES
    (trigger_seq.NEXTVAL, 'BEFORE СТРОКОВЫЙ ТРИГГЕР')
END classesBERowt;

CREATE [OR REPLACE] TRIGGER classesAFRow
AFTER UPDATE ON classes FOR EACH ROW
BEGIN
    INSERT INTO temp_table (num_col, char_col)
            VALUES (trigger_seq.NEXTVAL, 'AFTER СТРОКОВЫЙ ТРИГГЕР')
END classesAFRow;
```

Теперь выполним оператор UPDATE:

```
UPDATE classes
   SET num_credit =4
WHERE department IN ('AA','BB');
```

Этот оператор воздействует на 4 строки. Каждый из операторных триггеров выполняется один раз, а каждый из строковых – 4 раза.

Ограничения, налагаемые на триггеры

Тело триггера является блоком PL/SQL.

Любой оператор, выполнение которого разрешено в блоке PL/SQL, можно выполнить и в теле триггера при условии соблюдения следующих ограничений:

- ✔ В триггере нельзя задавать ни один из операторов управления транзакциями: COMMIT, ROLLBACK или SAVEPOINT.
- ✔ Срабатывание триггера является частью процесса выполнения активизирующего оператора, то есть частью той транзакции, которая охватывает и активизирующий оператор.
- ✔ Когда этот оператор завершается или откатывается, все выполненное триггером также завершается или откатывается.
- ✔ В процедурах и функциях, вызывающихся в теле триггера, также нельзя задавать какие-либо из операторов управления транзакциями.
- ✔ В теле триггера нельзя объявлять переменные с типами LONG и LONG RAW.
- ✔ Кроме того, в псевдозаписях :new и :old (см. ниже) нельзя ссылаться на столбцы типов LONG и LONG RAW таблицы, для которой определен триггер.

Из тела триггера можно обращаться не ко всем таблицам в зависимости от типа триггера и ограничений, накладываемых на таблицы.

Использование :old и :new в строковых триггерах

Строковый триггер срабатывает один раз для каждой строки, обрабатываемой активизирующим оператором.

Внутри триггера можно обращаться к строке, обрабатываемой в данный момент. Для этого служат две псевдозаписи - :old и :new.

Хотя синтаксически они рассматриваются как записи, фактически они записями не являются.

Поэтому их называют псевдозаписями.

Тип обеих псевдозаписей определяется:

***активизирующая_таблица%*ROWTYPE;**

Хотя :old и :new синтаксически рассматриваются в качестве записей типа *активизирующая_таблица%*ROWTYPE, в действительности они записями не являются. Псевдозаписи нельзя присваивать чему-либо целиком, можно только поля псевдозаписей.

:new модифицируется только в строковом триггере BEFORE, а :old никогда не модифицируется, а лишь считывается.

Активизирующий оператор	:old	:new
INSERT	Не определена – во всех полях NULL - значения	Значения, которые будут выведены после выполнения оператора
UPDATE	Исходные значения, содержащиеся в строке перед обновлением данных	Новые значения, которые будут введены после выполнения оператора
DELETE	Исходные значения, содержащиеся в строке перед ее удалением	Не определена – во всех полях NULL - значения

Доступ к значениям столбцов

Доступ к значениям столбцов в триггере осуществляется с помощью корреляционных имен:

:NEW.имя_столбца — новое значение;

:OLD.имя_столбца — старое значение.

В триггере INSERT имеют смысл только новые значения, для DELETE — только старые значения, для UPDATE — оба.

Пример тригерра BEFORE, срабатывающего на операторы INSERT и UPDATE, заполняющий поле ID в таблице Students значением, генерируемым последлвательностью trigger_seq.

```
CREATE [OR REPLACE] TRIGGER GenerateStudentId
    BEFORE INSERT OR UPDATE ON Students FOR EACH ROW
BEGIN
        SELECT student_seq.nextval
                INTO :new.ID
                FROM dual;
END GenerateStudentId;
```

Примеры триггеров:

1) CREATE TRIGGER deletecustomer
BEFORE DELETE ON customer
FOR EACH ROW
BEGIN
INSERT INTO customerhistory
VALUES (:old.id, :old.lastname, :old.firstname);
END deletecustomer;

2) CREATE TRIGGER timecheck
BEFORE UPDATE OR DELETE ON customer
BEGIN
IF to_char (sysdate, ‘HH24’) NOT BETWEEN 7 or 18 THEN
RAISE_APPLICATION_ERROR (-20101, ‘изменять запись о покупателе в это время не допускается’);
ENDIF
END timecheck;

Примеры триггеров:

```
3)  CREATE TRIGGER updatestockquantity
      AFTER INSERT OR DELETE OR UPDATE  OF quantity ON item
        FOR EACH ROW
  BEGIN
    IF inserting THEN
       UPDATE stock SET onhand = onhand - :new.quantity
            WHERE id = :new.stockid;
     ELSIF updating THEN
    IF :new.quantity > :old.quantity THEN
  UPDATE stock SET onhand = onhand – (:new.quantity - :old.quantity)
            WHERE id = :new.stockid;
   ELSE
  UPDATE stock SET onhand = onhand + (:old.quantity - :new.quantity)
            WHERE id = :new.stockid;
    ENDIF;
  ELSE     UPDATE stock SET onhand = onhand + :old.quantity
            WHERE id = :new.stockid;
ENDIF;
END updatestockquantity;
```

Параллельные архитектуры серверов баз данных

Три основные архитектурные направления:

- Симметричные многопроцессорные системы (SMP) - форма сильносвязанных многопроцессорных систем, разделяющих единую оперативную память и дисковую подсистему;
- Слабосвязанные многопроцессорные системы (кластеры) - совокупность компьютеров, объединенных в единую систему быстродействующей сетью и имеющих общую дисковую подсистему;
- Системы с массовым параллелизмом (MPP) - системы с сотнями и даже тысячами процессоров, имеющие многоуровневую структуру оперативной памяти

Группы требований, определяющих качества современной СУБД

- масштабируемость;
- производительность;
- возможность смешанной загрузки разными типами задач;
- обеспечение постоянной доступности данных (надежность или катастрофоустойчивость).

Масштабируемость - свойство вычислительной системы обеспечивать предсказуемый рост системных характеристик (число поддерживаемых пользователей, быстроты реакции, общей производительности) при добавлении к ней вычислительных ресурсов.

Факторы, влияющие на производительность СУБД:

- поддержка параллелизма (параллельный ввод/вывод, параллельные средства и утилиты администрирования, параллельная обработка запросов к базе данных)
- реализация многопотоковой архитектуры

Эволюция в области информационных систем все отчетливее направлена в сторону объединения задач: оперативной обработки транзакций (OLTP), поддержки принятия решений (DSS)

Группы требований, определяющих качества современной СУБД

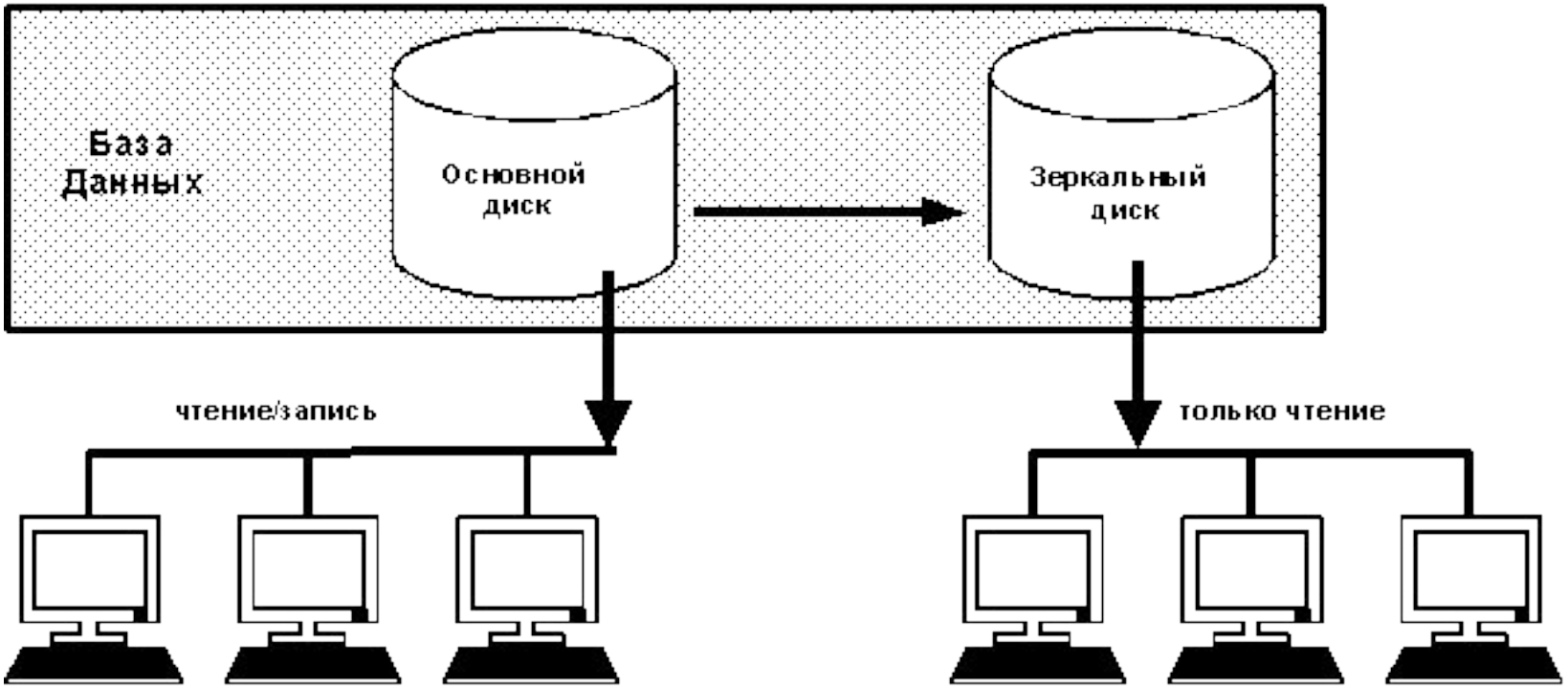
Постоянная доступность данных реализуется с помощью механизмов:

- оперативное администрирование;
- функциональная насыщенность СУБД.

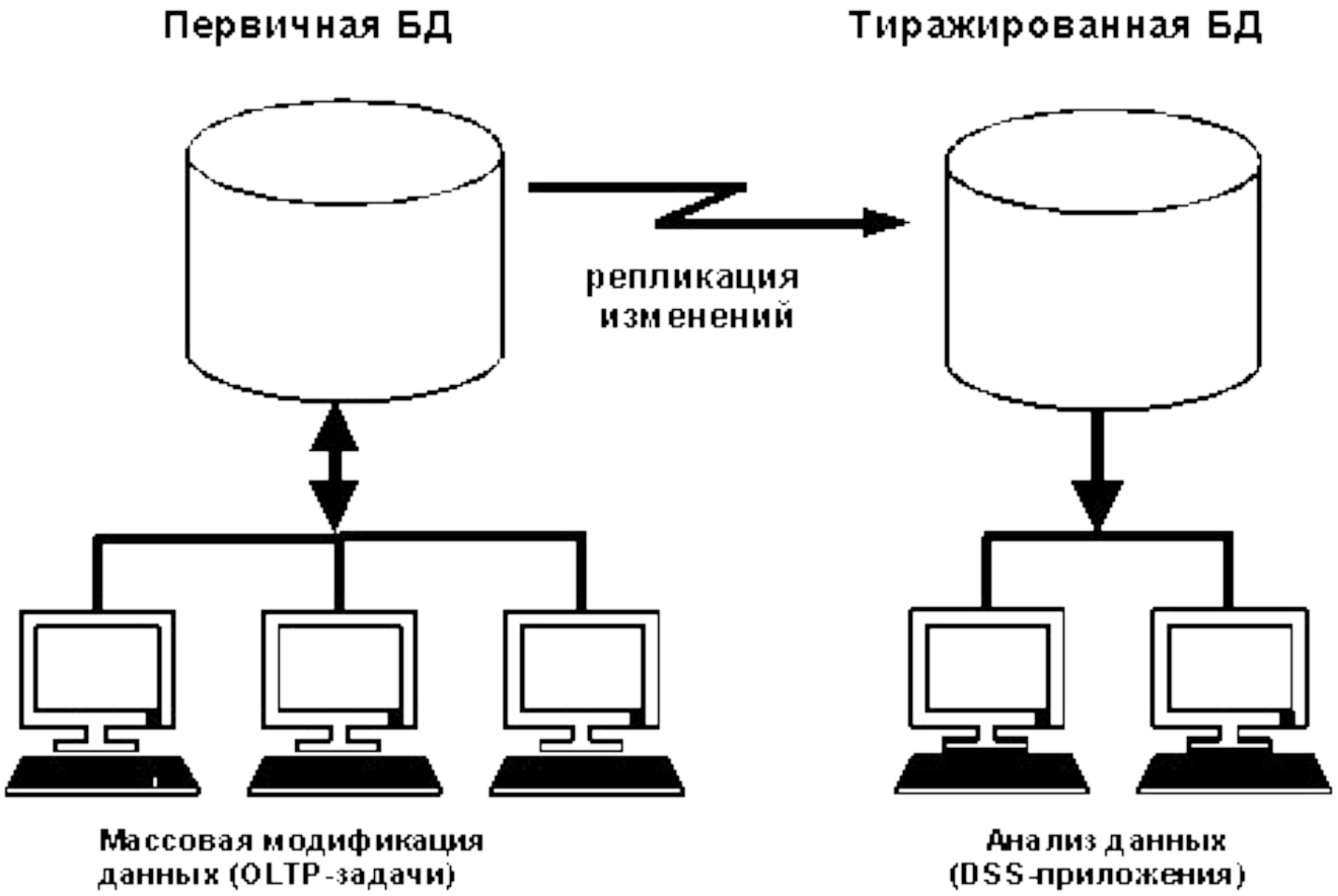
Утилиты администрирования призваны поддерживать бесперебойное функционирование СУБД, что подразумевает сведение к минимуму планируемых или сбойных простоев системы. Утилиты для пакетной загрузки/выгрузки данных, архивирования и восстановления, проверки целостности, реорганизации индекса должны эффективно выполняться в оперативном (on-line) режиме, без остановки СУБД, с использованием параллельных алгоритмов.

Управляемая избыточность данных обычно представлена в двух формах - программное зеркалирование (software mirroring) и тиражирование (replication) данных.

Зеркалирование (software mirroring)



Тиражирование (replication) данных



Распределенные системы баз данных

Ядром системы управления распределенными информационными ресурсами являются распределенная база данных и система управления распределенной базой данных.

Распределенная база данных – это совокупность логически взаимосвязанных баз данных, распределенных в компьютерной сети.

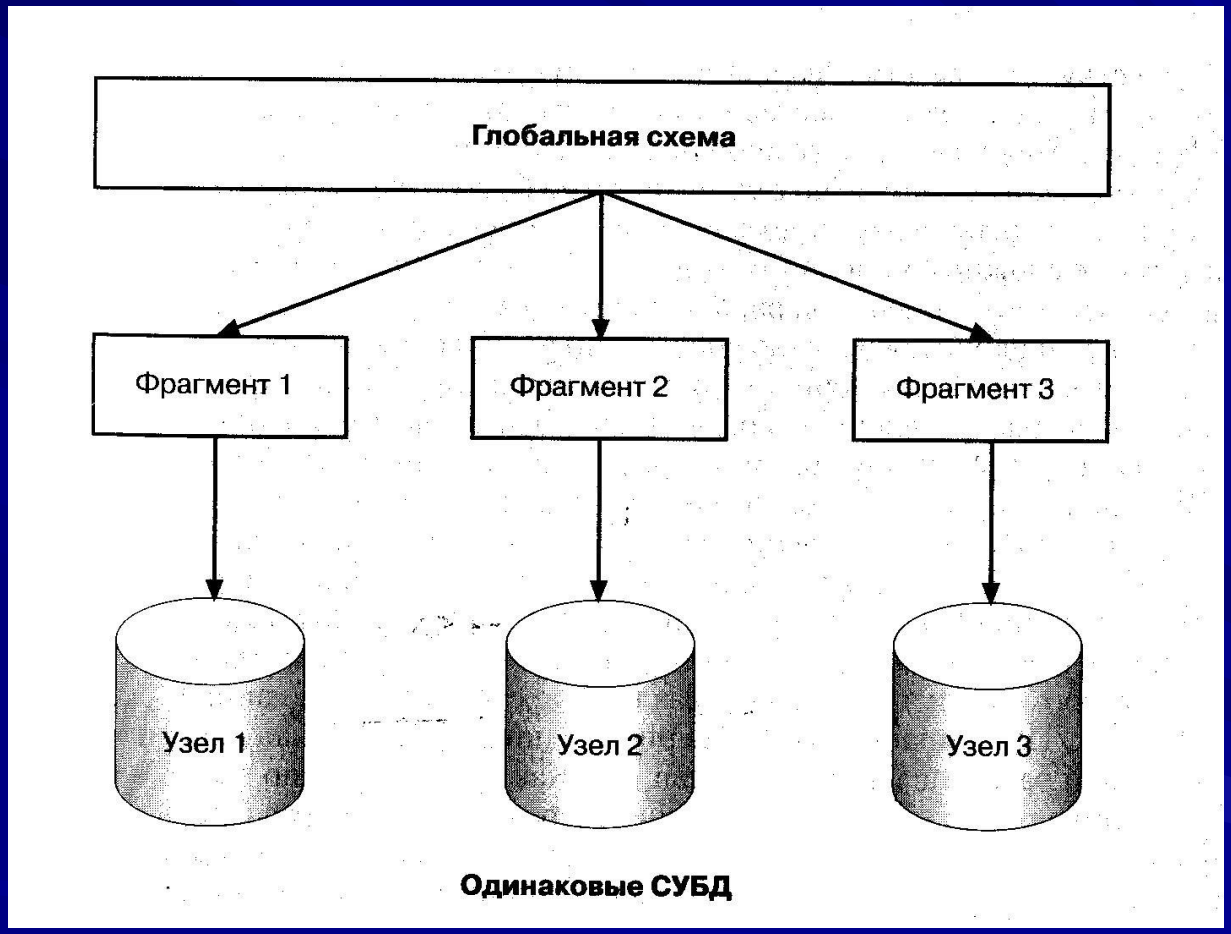
Система управления распределенной базой данных – программная система, которая обеспечивает управление распределенной базой данных и прозрачность ее распределенности для пользователей.

Распределение производится путем фрагментации или тиражирования

Правила К. Дейта для распределенных баз данных:

1. **Локальная автономность**
2. **Никакой конкретный сервис не должен возлагаться на какой-либо выделенный центральный узел.**
3. **Непрерывность функционирования.**
4. **Независимость от месторасположения.**
5. **Независимость от фрагментации.**
6. **Независимость от тиражирования.**
7. **Распределенная обработка запросов**
8. **Управление распределенными транзакциями.**
9. **Независимость от оборудования.**
10. **Независимость от операционных систем.**
11. **Независимость от сети.**
12. **Независимость от СУБД.**

Модели распределенных баз данных



Однородные системы, если СУБД –одинаковые, иначе – неоднородные системы

Фрагментация и тиражирование

Методы проектирования распределенных баз данных «сверху вниз» и «снизу вверх»

Проектирование «сверху вниз» аналогично проектированию централизованных баз данных:

- создание концептуальной модели базы данных;
- отображение ее в логическую модель данных;
- создание и настройка специфических для СУБД структур.

Однако при проектировании распределенной БД предполагается, что объекты не будут сосредоточены в одном месте, а распределятся по нескольким вычислительным системам.

Распределение проводится путем фрагментации и тиражирования.

Фрагментация означает декомпозицию объектов базы данных (например, таблицы) на несколько частей, которые размещаются на разных системах.

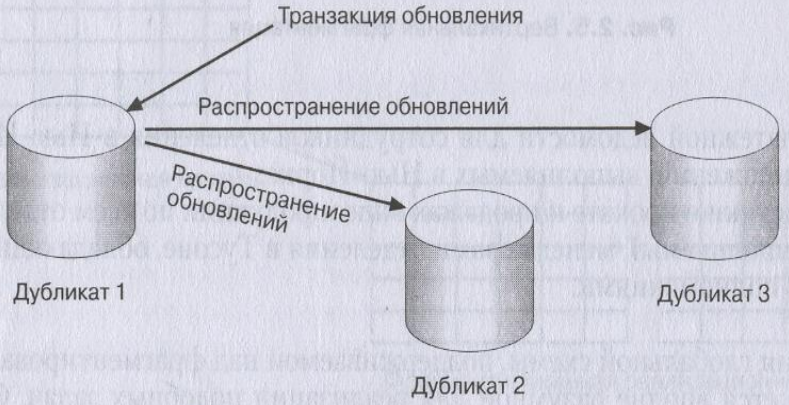
Существуют горизонтальная и вертикальная фрагментация (по строкам или по столбцам).

В любом случае поддерживается глобальная схема, позволяющая воссоздать из фрагментов логически централизованную таблицу или другую структуру.

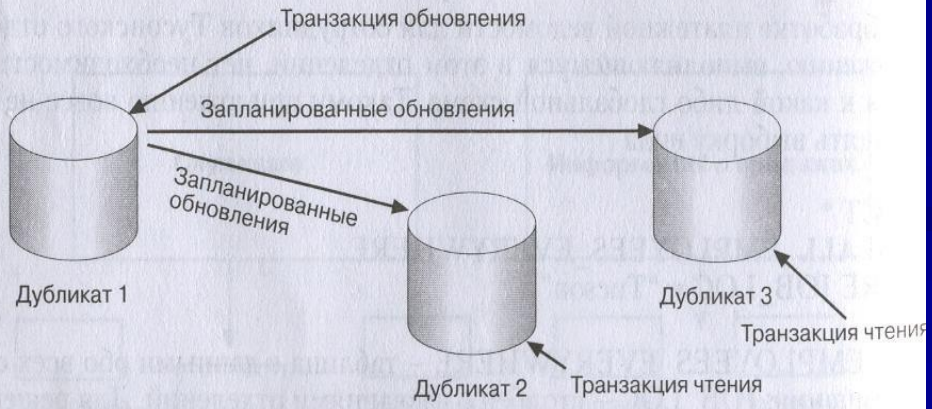
Тиражирование (или репликация) – создание дубликатов данных. Дубликаты (репликаты) – это множество различных копий некоторого объекта базы данных, для которых в соответствии с определенными правилами поддерживается синхронизация с главной копией.

Модели тиражирования данных

Одновременное обновление (с управлением параллелизмом)



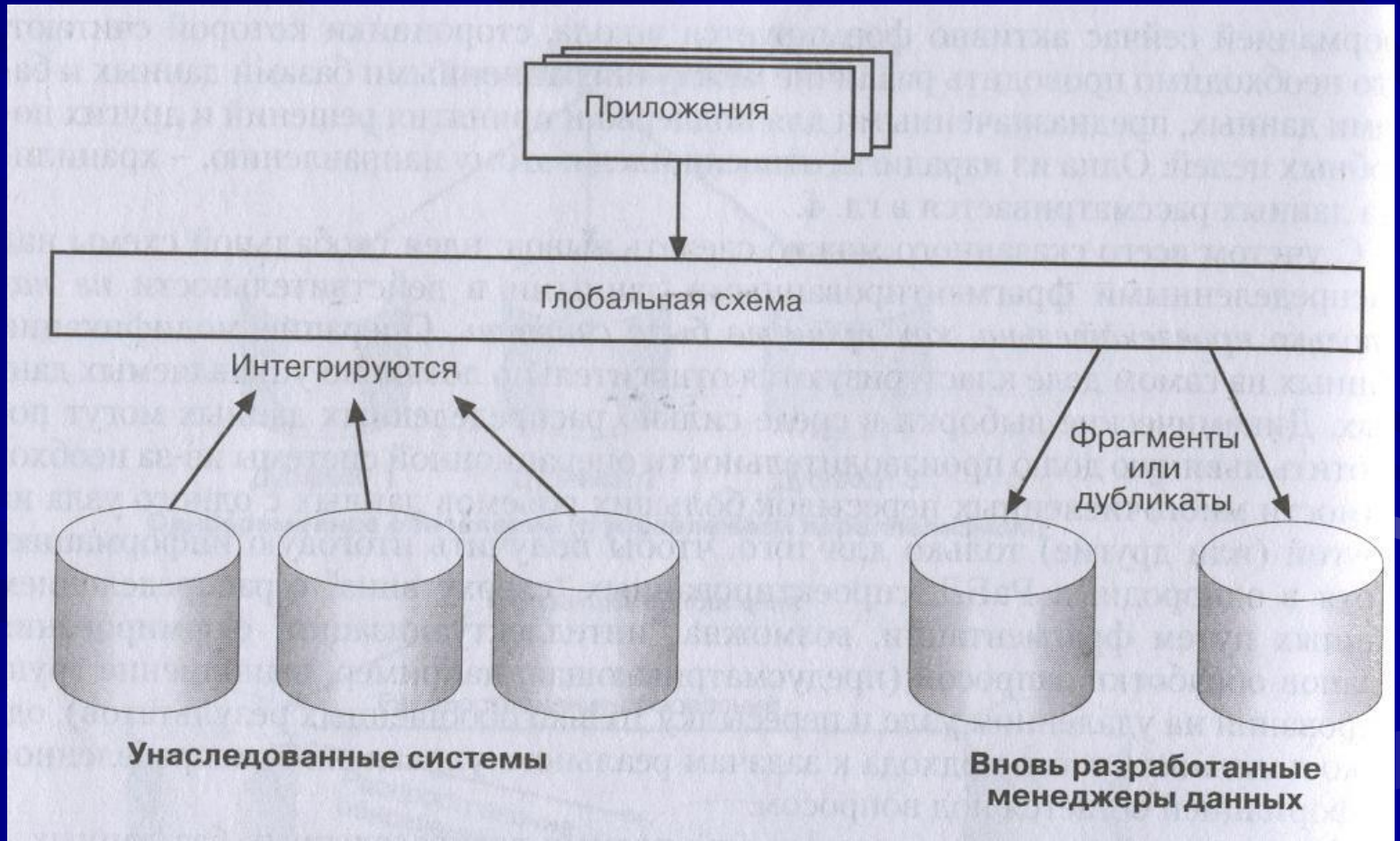
Распространенные обновления



Запланированная синхронизация дубликатов только для чтения

Интеграция распределенных баз данных «снизу-вверх»

Проектирование распределенных баз данных «снизу вверх» - объединение схем уже существующих БД, чтобы предоставить как новым, так и прежним приложениям доступ и к новым и к старым ресурсам данных (система мультибаз данных).



Средства защиты информации

Можно назвать следующие основные направления борьбы с потенциальными угрозами конфиденциальности и целостности данных:

- идентификация и проверка подлинности (аутентификация) пользователей;
- управление доступом к данным;
- механизм подотчетности всех действий, влияющих на безопасность;
- защита регистрационной информации от искажений и ее анализ;
- очистка объектов перед их повторным использованием;
- защита информации, передаваемой по линиям связи.

Для поддержания режима информационной безопасности особенно важны программно-технические меры, поскольку основная угроза компьютерным системам исходит от самих этих систем (сбои оборудования, ошибки программного обеспечения, промахи

Ключевые механизмы безопасности:

- идентификация и аутентификация;
- управление доступом (системные привилегии, объектные привилегии);
- механизм ролей (ROLE);
- Представления (VIEW);
- Триггеры;
- протоколирование и аудит;
- криптография;
- экранирование.

Некоторые определения

- Объект - пассивная единица информационного обмена, используется как синоним понятия данные.
- Субъект - активная единица информационного обмена, здесь выступает как синоним понятия процесс или приложение операционной системы.
- Аутентификация - верификация соответствия некоторого субъекта имеющейся априорной информации о нем.
- Авторизация - предоставление субъекту некоторых прав доступа к информационному объекту.

Аутентификация/авторизация при помощи паролей

1. Профили пользователей.
2. Профили процессов. Подобный метод реализован в технологии аутентификации Kerberos, где задачу аутентификации выполняет независимый (third-party) сервер, который содержит пароли как для пользователей, так и для конечных серверов
3. Комбинированные методы.

Инкапсуляция передаваемой информации в специальных протоколах обмена

1. **Инфраструктуры с открытыми ключами**. Использование подобных методов в коммуникациях основано на алгоритмах шифрования с открытым ключом. На этапе инициализации происходит создание пары ключей - открытого, который становится общеизвестным, и закрытого, имеющегося только у того, кто публикует открытый ключ. Суть алгоритмов шифрования с открытым ключом заключается в том, что операции шифрования и дешифрования производятся разными ключами (открытым и закрытым соответственно). Наиболее широко распространены следующие системы такого рода: ISO X.509 (в особенности его реализация для WWW,- Secure Socket Layer – SSL) - шифрование трафика транспортного уровня; Pretty Good Privacy (PGP) - общецелевая система шифрования с открытым ключом, наиболее широко используемая в системах электронной почты.
2. **Secure Shell protocol (ssh).** Протокол ssh используется для шифрования многих видов коммуникаций между удаленными системами (таких как копирование файлов или протокол X11). Данный протокол также использует шифрование с открытым ключом, но только на этапе установления соединений. Непосредственно транспортный трафик шифруется обычными алгоритмами: DES, 3DES, RC4 и др.
3. **Комбинированные методы**

Ограничение информационных потоков

1. **Firewalls.** Метод подразумевает создание между локальной и глобальной сетями специальных промежуточных серверов, которые инспектируют и фильтруют весь проходящий через них трафик сетевого/транспортного уровней. Более защищенная разновидность метода - это способ маскарада (masquerading), когда весь исходящий из локальной сети трафик посылается от имени firewall-сервера, делая локальную сеть практически невидимой.

2. **Proxy-servers**. При данном методе весь трафик сетевого/транспортного уровней между локальной и глобальной сетями запрещается полностью - попросту отсутствует маршрутизация как таковая, а обращения из локальной сети в глобальную происходят через специальные серверы-посредники. Очевидно, что при этом методе обращения из глобальной сети в локальную становятся невозможными в принципе. Очевидно также, что этот метод не дает достаточной защиты против атак на более высоких уровнях - например, на уровне приложения (вирусы, код Java и JavaScript).

Метки безопасности

Для реализации принудительного управления доступом с субъектами и объектами ассоциируются *метки безопасности*. Метка субъекта описывает его благонадежность, метка объекта - степень закрытости содержащейся в нем информации.

Согласно "Оранжевой книге", метки безопасности состоят из двух частей: уровня секретности и списка категорий. Уровни секретности, поддерживаемые системой, образуют упорядоченное множество:

- совершенно секретно;
- секретно;
- конфиденциально;
- несекретно.

Категории образуют неупорядоченный набор. Их назначение - описать предметную область, к которой относятся данные. В военной области каждая категория может соответствовать, например, определенному виду вооружений. Механизм категорий позволяет разделить информацию "по отсекам", что способствует лучшей защищенности. Главная проблема, которую необходимо решать в связи с метками, - это обеспечение их целостности.

Принудительное управление доступом

Принудительное управление доступом основано на сопоставлении меток безопасности субъекта и объекта.

Субъект может читать информацию из объекта, если уровень секретности субъекта не ниже, чем у объекта, а все категории, перечисленные в метке безопасности объекта, присутствуют в метке субъекта.

Описанный способ управления доступом называется принудительным, поскольку он не зависит от воли субъектов (даже системных администраторов). После того как зафиксированы метки безопасности субъектов и объектов, оказываются зафиксированными и права доступа.

Классы безопасности

Министерство обороны США ранжировали информационные системы по степени надежности. В "Оранжевой книге" определяется четыре уровня безопасности (надежности) - D, C, B и A.

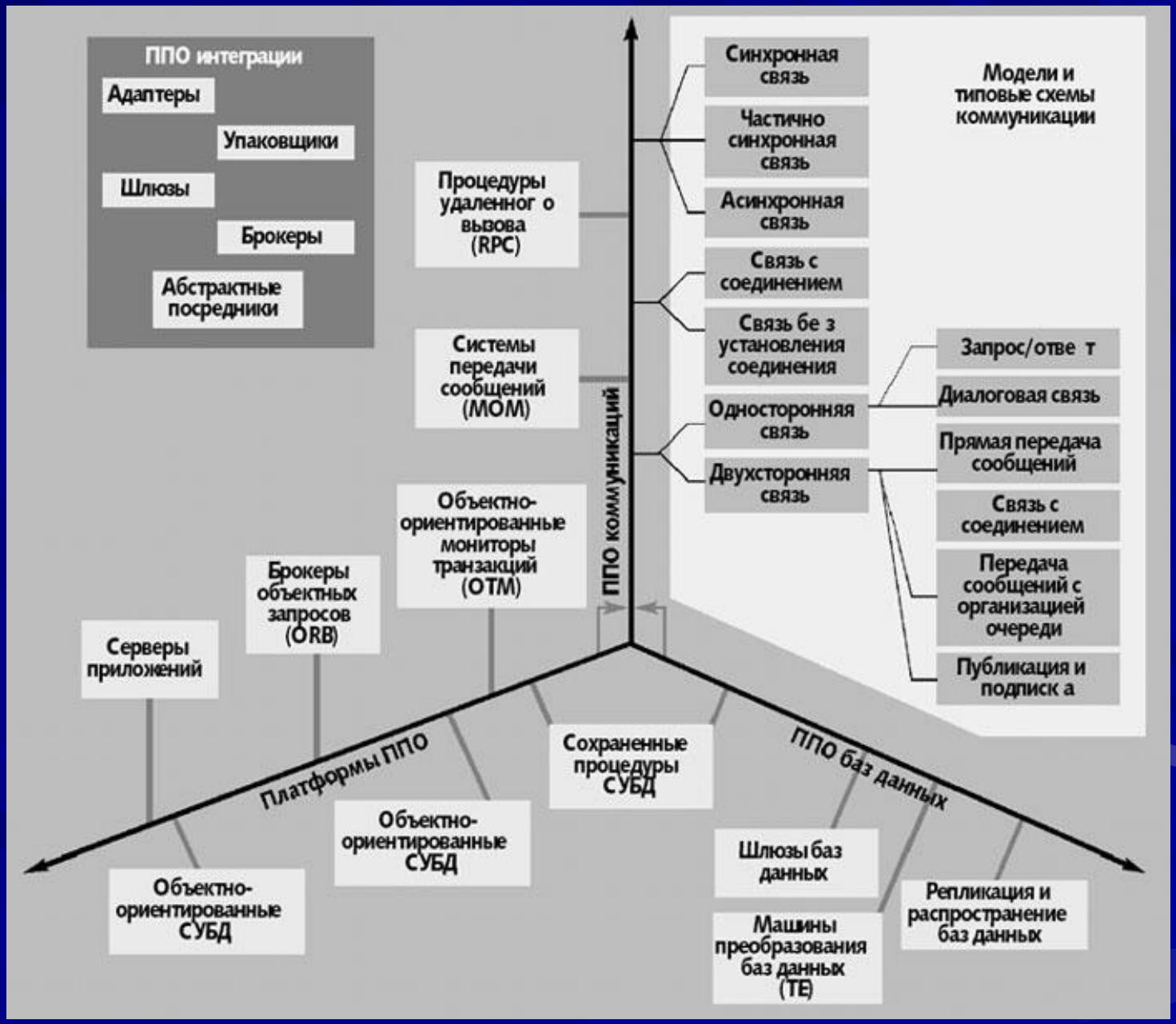
Промежуточное программное обеспечение (Middleware)

В распределенной неоднородной среде ППО играет роль «информационной шины», надстроенной над сетевым уровнем и обеспечивающей доступ приложения к разнородным ресурсам, а также независимую от платформ взаимосвязь различных прикладных компонентов.

Классификация ППО

- Три ключевых типа ППО - коммуникации, базы данных и платформные средства - условно показаны в виде трех осей.
- **ППО коммуникаций**, состоящее из процедур удаленного вызова (RPC - remote procedure call) и систем передачи и обработки сообщений (MOM - message oriented middleware), специализируется в обеспечении механизмов передачи информации между программами, выполняющимися на одном или разных компьютерах.
- **ППО баз данных** обеспечивает механизмы доступа к удаленным источникам данных.
- **Платформное ППО**, включающее серверы приложений, мониторы обработки транзакций (TPM - transaction processing monitor), объектно-ориентированные мониторы транзакций (OTM - object transaction monitor) и объектно-ориентированные брокеры запросов (ORB - object request broker), помимо простой коммуникации, поддерживает дополнительные службы: управление процессами и памятью, восстановление при ошибках, балансировку загрузки и обработку транзакций. Кроме того, оно преодолевает ограниченность конкретными типами баз данных, свойственных собственно ППО баз данных.

Типы ППО


ППО интеграции
Адаптеры
Упаковщики
Шлюзы
Брокеры
Абстрактные посредники
Процедуры удаленног о вызова (RPC)
Системы передачи сообщений (MOM)
Объектно-ориентированные мониторы транзакций (OTM)
Брокеры объектных запросов (ORB)
Серверы приложений
ППО коммуникаций
Синхронная связь
Частично синхронная связь
Асинхронная связь
Связь с соединением
Связь бе з установления соединения
Односторонняя связь
Двухсторонняя связь
Запрос/отве т
Диалоговая связь
Прямая передача сообщений
Связь с соединением
Передача сообщений с организацией очереди
Публикация и подписк а
Модели и типовые схемы коммуникации
Платформы ППО
ППО баз данных
Сохраненные процедуры СУБД
Объектно-ориентированные СУБД
Объектно-ориентированные СУБД
Шлюзы баз данных
Машины преобразования баз данных (TE)
Репликация и распространение баз данных

Промежуточное программное обеспечение баз данных

ППО баз данных обеспечивает доступ к локальному или удаленному ресурсу данных и включает некоторые низкоуровневые средства ППО коммуникаций, необходимые для связи клиента и сервера. Основная задача ППО баз данных - скрыть сложное расположение распределенного ресурса данных. ППО баз данных включает шлюзы, концентраторы, универсальные API-интерфейсы СУБД, процессоры преобразования данных, также как и инструментальные средства передачи изменений между несколькими экземплярами баз данных.

Шлюзы и концентраторы баз данных обеспечивают доступ на языке SQL к разнотипным источникам данных. Когда источники не поддерживают SQL, шлюзы транслируют SQL-запросы, полученные от приложения, в запросы, понятные целевой базе данных. Концентратор базы данных аналогичен шлюзу, но может иметь дело с несколькими источниками данных одновременно.

В зависимости от конкретного продукта преобразование выполняется либо на уровне API-интерфейса, либо на уровне протоколов коммуникаций, либо на обоих уровнях сразу.

Универсальный API-интерфейс - интерфейс к источнику данных, отрывающий приложение от определенной базы данных, обеспечивая единообразный интерфейс независимо от специфической архитектуры используемой СУБД. Три наиболее известных стандарта универсальных API-интерфейса СУБД: Open Database Connectivity (ODBC), Java Database Connectivity (JDBC) и Object Linking and Embedding Database (OLE DB).

Доступ к базам данных

- Системы прозрачного доступа к БД представляют собой наиболее развитый сектор рынка ППО. В простых двухзвенных моделях клиент-сервер, где несколько баз данных обслуживают ограниченное число пользователей настольных ПК, в роли встроенного MiddleWare (MW) доступа к данным могут выступать обычные ODBC-драйверы. Необходимость в более сложных решениях возникает в больших, разнородных многозвенных системах, где множество приложений в параллельном режиме осуществляет доступ к разнообразным источникам данных, включая СУБД и хранилища данных от различных поставщиков. В таких системах между клиентами и серверами баз данных размещается промежуточное звено – SQL-шлюз, который представляет собой набор общих API, позволяющих разработчику строить унифицированные запросы к разнородным данным (в формате SQL или с помощью ODBC-интерфейса). SQL-шлюз выполняет синтаксический разбор такого запроса, анализирует и оптимизирует его и выполняет преобразование в SQL-диалект нужной СУБД. MW этого типа реализует синхронный механизм связи, когда выполнение приложения, сделавшего запрос, блокируется до момента получения данных. Надо заметить, что синхронные принципы взаимодействия в распределенной среде, как правило, порождают проблемы масштабируемости системы.

Доступ к базам данных

- Использование MW доступа к БД широко применяется в корпоративных системах поддержки принятия решений (DSS), которые собирают и анализируют данные из множества разнородных источников и не требуют управления оперативными транзакциями.
- Рынок средств прозрачного доступа к базам данных практически не стандартизован – поставщики обычно создают свои частные решения и не обременены проблемами совместимости. Это можно объяснить тем, что приложение, использующее данный тип MW, извлекает информацию непосредственно из статического источника (хранилища данных), а не обращается за ней к другому прикладному модулю, возможно, от другого поставщика.

Архитектура ODBC(OPEN DATABASE CONNECTIVITY)

- SQL - приложение
  Администратор ODBC
  Драйверы ODBC для различных СУБД
  локальные или удаленные БД
- Основная идея:

  - все операции с базой данных идут через специальный программный слой, не зависящий от СУБД;

  - конфигурация ODBC для каждого источника данных (alias) определяет его драйвер и местоположение;

  - при изменении драйвера или местоположения необходимо изменить эти параметры в конфигурации
- Уровни драйверов ODBC: минимальный, базовый, расширенный.

- Существует 4 важных этапа (шага) процедуры запроса данных через ODBC API.
- *Шаг 1* - установление соединения. Первый шаг состоит в размещении указателей (handle) среды ODBC, которые выделяют оперативную память под ODBC драйверы и библиотеки. Затем происходит выделение памяти для указателей соединения, и соединение устанавливается.
- *Шаг 2* - выполнение оператора SQL. Выделяется указатель оператора, локальные переменные связываются со столбцами в SQL-выражении (это необязательное действие), и выражение представляется главному ODBC-драйверу для обработки.
- *Шаг 3* - извлечение данных. Перед извлечением данных возвращается информация о результирующем наборе, в частности, число столбцов в наборе. Исходя из этого числа, результирующий набор помещается в буфер записей, выполняется цикл его просмотра и содержимое каждого столбца помещается в соответствующую локальную переменную.
- *Шаг 4* - освобождение ресурсов.
- Технология ODBC разрабатывалась как общий, независимый от источников данных, способ доступа к данным. Применение технологии обеспечивает переносимость приложений в среду различных баз данных без переработки самих приложений. Технология ODBC уже стала промышленным стандартом, ее поддерживают практически все производители СУБД и средств разработки.

Недостатки реляционных СУБД

- Слабое представление сущностей реального мира
- Семантическая перегрузка
- Слабая поддержка ограничений целостности и корпоративных ограничений
- Однородная структура данных
- Ограниченный набор операций
- Трудности организации рекурсивных запросов
- Проблема рассогласования
- Другие проблемы РСУБД, связанные с параллельностью, изменениями схемы и слабыми средствами доступа

Манифест систем объектно-ориентированных баз данных

Обязательные свойства: золотые правила

Система объектно-ориентированных баз данных должна удовлетворять двум критериям: она должна быть СУБД и при этом являться объектно-ориентированной системой, т.е. в максимально возможной степени находиться на уровне современных объектно-ориентированных языков программирования.

Первый критерий означает пять свойств: стабильность (persistence), управление вторичной памятью, параллелизм, восстанавливаемость и средства обеспечения незапланированных запросов.

Второй означает восемь свойств: сложные объекты, идентифицируемость объектов, инкапсуляцию, типы или классы, наследование, перекрытие методов совместно с поздним связыванием, расширяемость и вычислительную полноту.

Необязательные возможности

Множественное наследование, проверка и вывод типов, распределенность, проектные транзакции (протяженные транзакции или вложенные транзакции), версии

Преимущества и недостатки ООСУБД

Преимущества:

- Улучшенные возможности моделирования
- Расширяемость
- Устранение проблемы несоответствия
- Более выразительный язык запросов
- Поддержка эволюции схемы
- Поддержка долговременных транзакций
- Применимость для сложных специализированных приложений баз данных
- Повышенная производительность

***Недостатки*:**

- Отсутствие универсальной модели данных
- Недостаточность опыта эксплуатации
- Отсутствие стандартов
- Влияние оптимизации запросов на инкапсуляцию
- Влияние блокировки на уровне объекта на производительность
- Сложность
- Отсутствие поддержки представлений
- Недостаточность средств обеспечения безопасности

Объектная модель данных

В соответствии со стандартом ODMG 2.0 объектная модель данных характеризуется следующими свойствами.

- ▪ Базовыми примитивами являются объекты и литералы. Каждый объект имеет уникальный идентификатор, литерал не имеет идентификатора.
- ▪ Объекты и литералы различаются по типу. Все элементы одного типа имеют одинаковый диапазон изменения состояния (множество свойств) и одинаковое поведение (множество определенных операций). Объект, на который можно установить ссылку, называется экземпляром; он хранит определенный набор данных.
- ▪ Состояние объекта определяется набором значений, реализуемых множеством свойств. Этими свойствами могут быть атрибуты объекта или связи между объектом и одним или несколькими другими объектами.
- ▪ Поведение объекта определяется набором операций, которые могут быть выполнены над объектом или самим объектом. Операции могут иметь список входных и выходных параметров строго определенного типа. Каждая операция может также возвращать типизированный результат.
- ▪ База данных хранит объекты, позволяя совместно использовать их различным пользователям и приложениям. База данных основана на схеме данных, определяемой языком определения данных, и содержит экземпляры типов, определенных схемой.

Объект, тип

Каждый тип имеет внешнюю спецификацию и одну или несколько реализаций. Спецификация определяет внешние характеристики типа: пользователю для работы с объектом предоставляется набор операций и набор атрибутов объекта, при помощи которых можно работать с реальными экземплярами. Реализация определяет внутреннее содержание объектов, например операции.

Тип также является объектом. Поддерживается иерархия супертипов и подтипов, реализуя стандартный механизм объектно-ориентированного программирования — наследование.

ООСУБД обслуживает множество баз данных, каждая из которых содержит определенное множество типов.

Идентификатор объекта

Как это следует из модели данных, каждый объект в базе данных уникален. Существует несколько подходов для идентификации объекта. Самый простой — присвоить ему уникальный номер (OID — object identificator) в базе и никогда больше не повторять этот номер, даже если предыдущий объект с таким номером уже удален. Недостаток такого подхода состоит в невозможности перенести объекты в другую базу без потери связности между ними. Решение этой проблемы заключается в использовании составного идентификатора. Например, в Versant идентификатор OID имеет формат xxxxxxxx:yyyyyyyy, где xxxxxxxx — идентификатор базы данных, yyyyyyyy — идентификатор объекта в базе. Составленный таким образом OID позволяет переносить объекты из базы в базу без потери связи между объектами или без удаления объектов с перекрывающими номерами.

Идеальный вариант — использование OID, состоящего из трех частей: номер базы, номер класса, номер объекта. Однако и при этом остается вопрос о том, как обеспечить уникальность номеров баз и классов на глобальном уровне — при использовании ООСУБД на различных платформах, в разных городах и странах.

Новые типы данных

Одним из принципиальных отличий объектных баз данных от реляционных является возможность создания и использования новых типов данных. Концептуально объект характеризуется поведением и состоянием. Определение типа заключается в определении поведения, т.е. операций, которые могут быть выполнены объектом или над состоянием объекта — набором атрибутов определенных типов (атрибут может иметь любой объявленный в базе тип). Важная особенность ООСУБД состоит в том, что создание нового типа не требует модификации ядра базы и основано на принципах объектно-ориентированного программирования: инкапсуляции, наследовании, перегрузке операций и позднем связывании.

Функционирование базы основано на схеме данных, которая может быть как первичной для создания классов или вторичной, выделяемой из созданных на языке программирования (C++) классов и загружаемой в базу. Язык ODL разработан ODMG как универсальный язык описания объектов. Для целей разработки предусмотрены элементы расширения классических объектных языков C++, Smalltalk, Java, позволяющих описать структуру объектов, их связи и типы связей.

Оптимизация ядра СУБД

Ядро ООСУБД оптимизировано для операций с объектами. Естественными операциями для него являются кэширование объектов, ведение версий объектов, разделение прав доступа к конкретным объектам. Ядро объектно-реляционной СУБД остается реляционным, а «объектность» реализуется в виде специальной надстройки. Как следствие, ООСУБД свойственно более высокое быстродействие на операциях, требующих доступа и получения данных, упакованных в объекты, по сравнению с реляционными СУБД, для которых необходимость выборки связных данных ведет к выполнению дополнительных внутренних операций

Язык СУБД и запросы

Общепризнанны две группы вариантов языков запросов.

- Язык OQL (Object Query Language) для объектных баз данных. Объектно-реляционные СУБД используют различные варианты объектных расширений SQL.
- Вторая группа языков запросов базируется на XML. Собирательное название языков этой группы — XML QL (или XQL). Они могут применяться в качестве языков запросов в объектных и объектно-реляционных базах данных.

Транзакции

Короткие транзакции характеризуются малым временем выполнения; они могут существовать только в рамках сеанса работы с ООСУБД. Все изменяемые объекты блокируются, а после принятия транзакции разблокируются, изменения же записываются в базу данных.

Длинные транзакции предназначены для увеличения производительности при групповой работе. Можно создавать персональные и групповые базы. Пользователи работают со своей базой, а объекты из нее синхронизируются с групповой базой данных. Пользователь, начав длинную транзакцию, отмечает объекты, с которыми предстоит работать в групповой базе данных (операция «поставить на контроль» — check out). Эти объекты копируются в его персональную базу, а в групповой базе блокируются, причем блокировать их можно как на запись, так и на чтение. В групповой базе создается объект, содержащий все данные о длинных транзакциях. В случае повреждения групповой базы или физического отключения сервера групповой базы пользователь сможет продолжать работу с объектами в своей персональной базе, а после восстановления групповой базы — синхронизировать объекты. Перед завершением длинной транзакции пользователь должен поместить все измененные объекты обратно в основную базу (операция «зарегистрировать» — check in). После этого объекты копируются в основную базу, а блокировка снимается. В случае аварийного завершения длинной транзакции все изменения будут потеряны.

Вложенные транзакции по принципу функционирования аналогичны коротким. В процессе выполнения одной транзакции формируются другие. Если в текущем сеансе работает один процесс, то создается стек, а если несколько процессов — дерево транзакций.

Блокировки

Назначение блокировок — гарантировать монопольность использования объекта конкретным пользователем с целью предотвращения одновременного изменения данных.

Короткие блокировки (short lock) предназначены для обеспечения последовательного доступа к данных при многопользовательском режиме работы.

Продолжительные блокировки (persistent lock) обеспечивают блокирование объектов на продолжительное время — часы, дни, недели. Применяются совместно с длинными транзакциями.

При этом объект может быть заблокирован несколькими способами:

- с исключением снятия другим процессом (hard lock);
- с возможностью снятия другим процессом (soft lock);
- по конкретным операциям.

Перемещение объектов

Миграция объектов: постоянное их перемещение, например в другую базу данных. В качестве примера можно привести перемещение объектов из базы оперативных данных в базу данных архивного назначения.

Постановка на контроль (check out): копирование объектов в персональную базу данных при выполнении длинной транзакции.

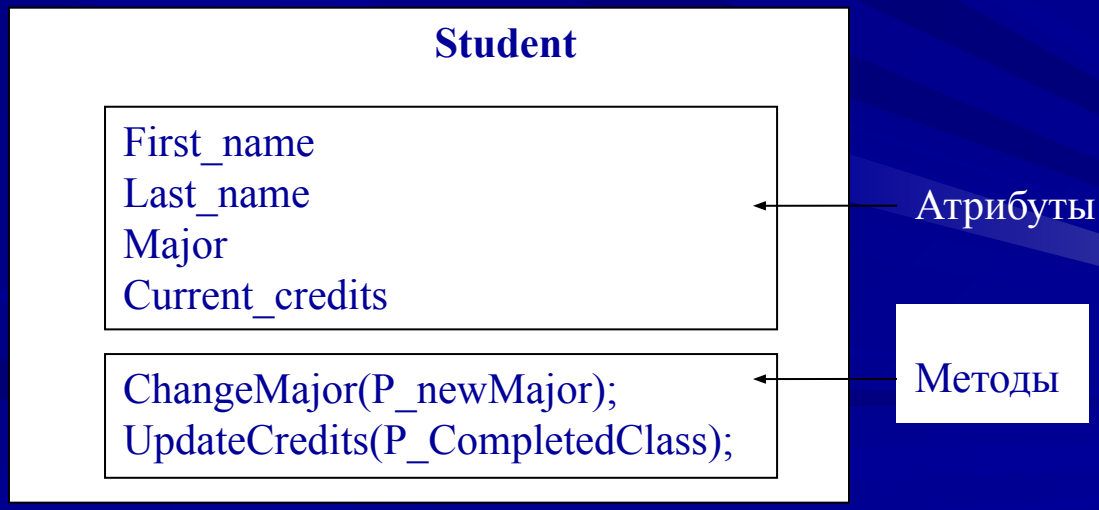
Регистрация объектов (check in): копирование объектов в групповую базу данных из персональной при выполнении длинной транзакции.

Ведение версий

В «Манифесте объектно-ориентированных баз данных», поддержка множественных версий объектов отнесена к необязательным характеристикам ООСУБД. Однако большинство современных ООСУБД поддерживает версионность, что способствует повышению надежности информационной системы в целом.

Основы объектно-ориентированного программирования

Рассмотрим объект Student. Студент имеет атрибуты, т.е. свойства: имя (first_name), фамилию (last_name), профилирующую дисциплину (major), и текущее количество полученных зачетов (current_credits). Кроме того на рисунке указаны операции, влияющие на эти атрибуты: ChangeMajor (изменяет профилирующую дисциплину) и UpdateCredits (обновляет сведения о зачетах студента). Эти операции называются методами (methods). Объекты взаимодействуют между собой вызывая конкретные методы.



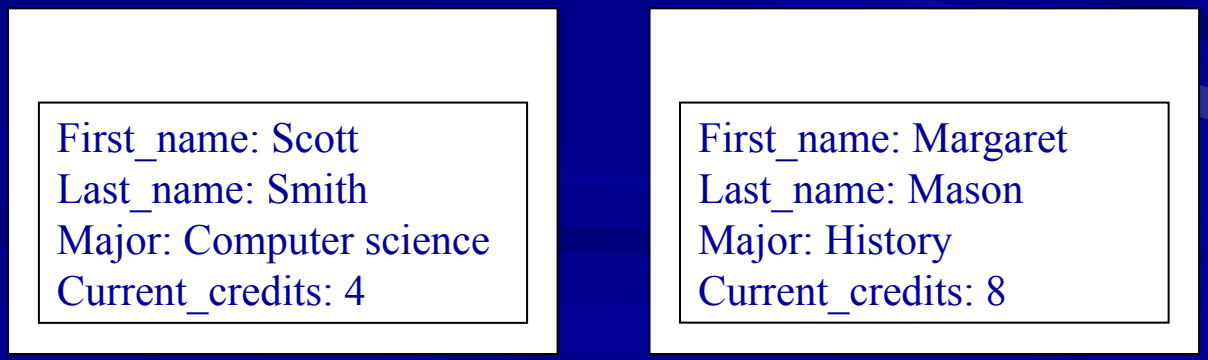
Объект Student

Абстракция

Как было показано на примере программных модулей, атрибуты и методы объекта достаточно точно реализуют абстрактное представление данных и процедур. В идеале клиент, использующий данный объект, управляет атрибутами только через методы.

Объекты и экземпляры объектов

Следует отметить различие, существующее между объектным типом и экземпляром этого типа. В системе может быть только один объектный тип с конкретным именем, но много его экземпляров. Экземпляр объекта похож на переменную: каждый экземпляр имеет свою область памяти и, как следствие, собственную копию атрибутов объекта. К примеру, на рис. 2 показано два экземпляра объектного типа **StudentObj**. Они аналогичны двум различным записям PL/SQL, объявленным с одним и тем же типом.



Экземпляры типа StudentObj

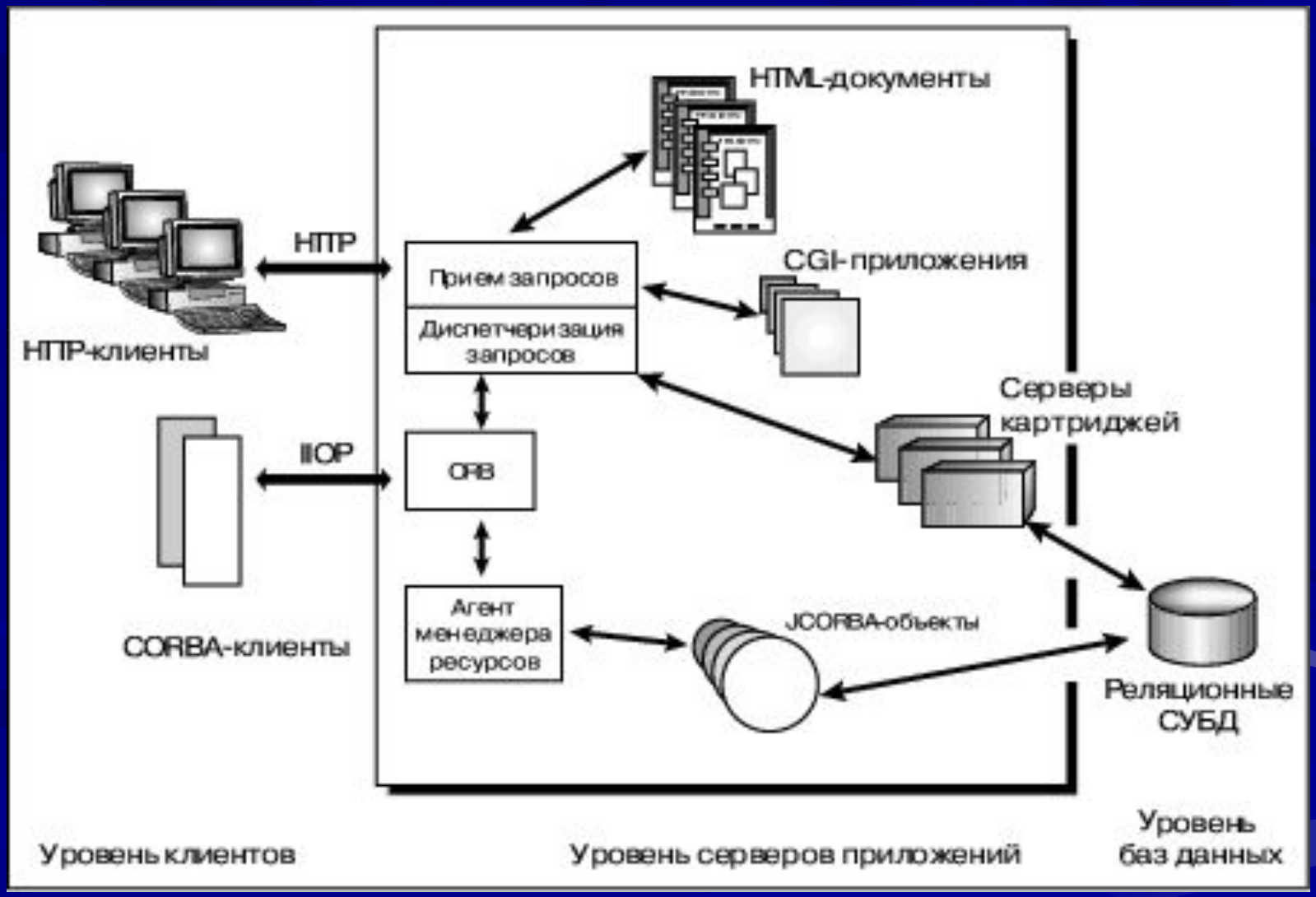
Объектно-реляционные базы данных

В настоящее время применяется множество объектно-ориентированных языков программирования, а том числе C++ и Java. Такие языки дают возможность описывать объекты и манипулировать ими, однако имеют существенный недостаток – они не обеспечивают надежного и корректного хранения и считывания объектов. Тут то и нужны объектно-реляционные базы данных, подобные Oracle8. Система Oracle8 создана для хранения объектных данных и для работы с ними. Управление объектными данными аналогично управлению реляционными данными и осуществляется с помощью языка SQL, выступающего в роли средства взаимодействия с базами данных. В объектно-реляционной базе данных язык SQL (и PL/SQL) используется для манипулирования как реляционными, так и объектными данными. Кроме того, Oracle8 обеспечивает:

- Эффективное управление транзакциями;
- Надежное резервное копирование и восстановление информации;
- Высокопроизводительную обработку запросов;
- Блокирование данных;
- Параллельность работы пользователей;
- Расширяемость самой системы.

Объединение объектов с реляционной моделью даст отличные результаты – эффективность и надежность реляционной базы данных соединятся с гибкостью и средствами моделирования объектной структуры.

Архитектура Oracle Application Server



СУБД Oracle9*i*

СУБД Oracle9*i* быстро превратилась в СУБД для всех типов данных – от простых до сложных. Мультимедийные типы данных, такие, как изображения, карты, видео- и аудио- клипы, редко обрабатывались неспециализированным программным обеспечением. Но в настоящее время многие веб-приложения требуют от своих серверов баз данных управления такими данными. Иные программные решения были также необходимы для хранения данных, которыми оперируют:

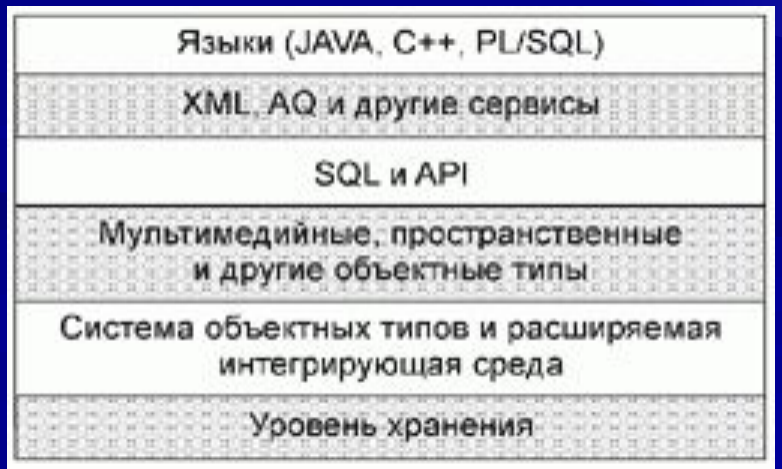
- финансовые инструменты;
- технические диаграммы;
- молекулярные структуры.

Для удовлетворения этих потребностей сервер баз данных Oracle9*i* предоставляет объектно-реляционную технологию, которая обеспечивает простые методы разработки, развертывания и управления приложениями, оперирующими со сложными данными.

Объектно-реляционная архитектура СУБД Oracle9*i*

Сервер Oracle9*i* с объектно-реляционной технологией может быть "подогнан" разработчиками для создания их собственных специфических для области применения (application-domain-specific) типов данных. СУБД Oracle9*i* ™ была расширена для поддержки полных возможностей объектного моделирования, включая наследование (inheritance) и многоуровневые коллекции (multi-level collections), а также эволюции типов данных (type evolution). Например, можно создать новые типы данных, представляющие клиентов (customers), финансовые портфели (financial portfolios), фотографии и телефонные сети – и, тем самым, обеспечить, чтобы ваши приложения баз данных оперировали абстракциями, свойственными вашей предметной области (application domain).

Кроме того, весьма желательно интегрировать эти новые типы с сервером баз данных настолько тесно, насколько это возможно, чтобы они обрабатывались наравне со встроенными типами данных, такими, как NUMBER или VARCHAR.



Объектно-ориентированная разработка приложений

СУБД Oracle9*i* предлагает большой набор интерфейсов прикладного программирования (API), реализующих связывания для различных языков. Для Java и PL/SQL предлагается "прямая" (native) поддержка внутри самой СУБД с тесной интеграцией между системой объектно-реляционных типов и хранимыми процедурами, написанными на Java или PL/SQL. Используя объектно-реляционную среду, можно хранить данные XML и эффективно манипулировать ими, индексировать их и эффективно обрабатывать запросы. Можно также поддерживать отображение между типами языка SQL и клиентских языков программирования (Java и C++), чтобы обеспечить "бесшовный" доступ к экземплярам типов данных SQL из приложений, написанных на Java или C++.

Индустриальные стандарты для разработки объектно-ориентированных приложений:

- UML (Unified Modeling Language) – унифицированный язык моделирования для объектно-ориентированного анализа и проектирования;
- стандарт объектно-реляционных баз данных SQL:1999;
- стандарты языков объектно-ориентированного программирования Java и C++.

Спецификации UML определяют стандартные конструкции для описания объектно-ориентированного программного обеспечения как объектной модели.

В 2003 г. консорциум OMG принял новую версию этого стандарта – UML 2.0.

Объектные типы данных

Объектный тип данных — это тип, определяемый пользователем, и задающий как структуру (атрибуты), так и поведение (методы) объектов.

Разделение интерфейса и реализации относится к числу общих мест объектного подхода. Описание объектного типа состоит из двух частей. В интерфейсной декларируются атрибуты объектов и заголовки методов (процедур и функций). В теле типа приводится реализация методов.

Две части описания объектного типа данных

Объектные типы

Корпорация Oracle расширила SQL, чтобы позволить пользователям:

- определять свои собственные типы (которые представляют их бизнес-объекты) и связи (например, наследование и агрегирование) между этими определяемыми пользователями типами;
- хранить их экземпляры (то есть, объекты) в базе данных (либо в столбцах таблиц, либо как сами таблицы);
- запрашивать, вставлять и изменять эти экземпляры.

Бизнес-объект:

- может содержаться внутри другого бизнес-объекта;
- на него может ссылаться другой объект (используя конструкцию REF);
- к нему можно получить доступ;
- с ним можно манипулировать как с коллекциями (collections) или наборами (sets), используя структуры, называемые массивами переменной длины (VARRAYS) и вложенными таблицами (Nested Tables).

Пользователи могут определять операции над бизнес-объектами как методы (methods) определяемых пользователями типов. Методы могут быть реализованы как хранимые процедуры на языках Java или PL/SQL. Объекты также обладают глобально уникальными идентификаторами, называемыми объектными идентификаторами (Object ID), которые могут быть использованы для поддержки ссылок между объектами.

СУБД Oracle9*i* позволяет пользователям рассматривать объектные данные как реляционные. Например, пользователи могут использовать SQL для запросов объектных данных точно так же, как для запросов реляционных данных. Пользователи могут получать доступ к объекту, используя операторы SQL DML, к его атрибутам и методам, используя расширенные выражения путей (например, объект.атрибут). Они могут также использовать SQL для выполнения явных соединений (explicit joins) объектов в таблицах. Кроме того, Oracle9*i* позволяет пользователям выполнять неявные соединения (implicit joins) объектов, путем обхода (traversing) или навигации по ссылкам от одного объекта к другому. Объекты можно индексировать, применяя методы MAP или ORDER для преобразования их в скалярные значения, которые затем могут быть индексированы.

Объектно-реляционные конструкции СУБД Oracle9*i* весьма близки к реляционным, которые хорошо знакомы пользователям СУБД Oracle. Например, ссылка REF очень похожа на внешний (foreign) ключ, методы – это хранимые процедуры (которые могут быть написаны на языках Java, PL/SQL или C/C++), модели безопасности и транзакций, оперирующие с объектными типами, являются точно такими же, как и модели, определенные для реляционных таблиц.

Наследование

Наследование типов (Type inheritance) – это фундаментальная концепция в любой объектно-ориентированной системе. Наследование типов позволяет совместно использовать похожие свойства различных типов, а также расширять их характеристики.

Во многих объектно-ориентированных приложениях объекты организованы в типы, а типы – в иерархии типов. Эмпирически вполне достаточно организовать иерархии типов в виде набора деревьев. Тем самым, простого наследования достаточно для поддержки организации типов в большинстве приложений. Java – это объектно-ориентированный язык программирования, поддерживающий простое наследование. С помощью простого наследования тип может расширять один супертип (наследовать от одного супертипа). Такой тип, называемый подтипом (subtype), наследует все атрибуты и методы своего супертипа (supertype). Подтипу можно также добавлять новые атрибуты и методы или переопределять унаследованные методы. СУБД Oracle поддерживает модель простого наследования.

Иерархия типов

Корневой тип иерархии создается с помощью оператора CREATE TYPE, в котором должен быть указано ключевое слово NOT FINAL (нетерминальный).

```
CREATE TYPE Person_t AS OBJECT(
name VARCHAR2(100),
dob DATE,
MEMBER FUNCTION age() RETURN number,
MEMBER FUNCTION print() RETURN varchar2) NOT FINAL;
```

Под нетерминальным типом может быть создан подтип. Он наследует все атрибуты и методы от своего супертипа. В нем можно добавить новые атрибуты и методы и/или переопределить унаследованные методы.

```
CREATE TYPE Employee_t UNDER Person_t(
salary NUMBER,
bonus NUMBER,
MEMBER FUNCTION wages() RETURN number,
OVERRIDING MEMBER FUNCTION print() RETURN varchar2);
```

Типы-коллекции

Коллекции – это типы данных SQL, составляющие элементы которых представляют собой множественные элементы. Каждый элемент или значение для коллекции обладает тем же самым подстановочным типом данных. В Oracle предусмотрено два типа коллекций – массивы переменной длины (Varrays) и вложенные таблицы (Nested Tables).

Массив переменной длины содержит переменное число упорядоченных элементов. Типы данных VARRAY могут быть использованы для столбцов таблиц или атрибутов объектных типов.

С помощью Oracle SQL можно создавать указанные выше типы таблиц. Они могут использоваться как вложенные таблицы для реализации семантики неупорядоченной коллекции. Так же как и VARRAY, типы вложенных таблиц могут быть использованы для столбцов таблиц или атрибутов объектных типов.

Ссылочные типы

Если вы создаете объектную таблицу или объектное представление в СУБД Oracle9*i*, то можно получить ссылку (или указатель базы данных, *pointer*) на соответствующий объект-строку (row object). Ссылки важны для моделирования связей и навигации по экземплярам объектов, в частности, в приложениях на стороне клиента.

Большие объекты

СУБД Oracle9*i* предоставляет типы LOB (large object, большой объект) для решения проблем хранения изображений, видеоклипов, документов и других видов неструктурированных данных. Большие объекты хранятся таким образом, чтобы было оптимизировано использование пространства памяти и обеспечен эффективный доступ к ним. Конкретнее, большие объекты состоят из указателей (locators) и связанных с ними двоичных и/или символьных данных. Указатели этих объектов хранятся в строках таблиц вместе со значениями других столбцов.

Если применяются внутренние большие объекты (BLOB, CLOB и NCLOB), их данные размещаются в отдельной области хранения.

Для внешних же объектов (BFILE) их данные хранятся вне базы данных в файлах операционной системы.

Связывания для языков программирования

Полная поддержка объектно-реляционной системы типов Oracle доступна в связываниях для ряда языков программирования, включая PL/SQL, Java и C/C++. К экземплярам типов можно получить доступ, и с ними можно манипулировать через интерфейсы прикладного программирования, такие, как JDBC (Java DataBase Connectivity) и OCCI (Oracle C++ Call Interface). Корпорация Oracle предоставляет также инструменты, подобные утилите JPublisher и транслятору объектных типов Object Type Translator (OTT), для отображения иерархий объектных типов в языки Java и C++. Кроме того, в средах этих языков также поддерживается подстановочность экземпляров и ссылок REF.

В СУБД Oracle 9i созданы новые и усовершенствованы существовавшие ранее механизмы обеспечения надежности и масштабируемости (Real Application Cluster, Logical Standby, FlashBack). Включена поддержка XML. В сервер Oracle интегрированы средства поддержки OLAP и добычи данных. Появился механизм Oracle Streams. Усовершенствованы средства управления, самонастройки и настройки.

Предусмотрена поддержка архитектуры IA–64.

Надежность и масштабируемость

Real Application Cluster (RAC)

Достижением Oracle 9i в области обеспечения высокой надежности и масштабируемости были средства поддержки кластеризации. Компонент RAC позволяет повысить надежность (при выходе из строя одного из узлов система продолжает функционировать), увеличивает масштабируемость (пользователи одной базы данных и одного приложения «размазываются» по всем узлам кластера), позволяет постепенно наращивать мощность системы, не останавливая ее работу.

Механизм Logical Standby

В Oracle 9i реализован механизм Logical Standby. Его отличие от физического заключается в том, что передаваемые в резервный центр изменения предварительно преобразуются в операторы SQL, причем процесс восстановления не блокирует работу базы данных в режиме чтения других пользователей.

Поддержка XML, дуализм XML/SQL

Сервер Oracle поддерживает не только реляционную, объектную, многомерную модель данных, но и XML. Поддерживаются XML-схемы и XML-объекты: таблицы с типом XMLType и колонки типа XMLType.

Реляционные и XML-данные сосуществуют в одной универсальной модели. С XML-данными можно работать посредством языков SQL и Java, а с реляционными — через XML-интерфейсы, например, через XPath. Поскольку из SQL можно работать с XML-данными и их частями, то теперь легко построить обычный индекс по реквизиту, содержащемуся в XML-файлах и быстро находить нужные файлы. Можно построить реляционное представление (View), колонками которого будут реквизиты XML-файлов и далее работать с этим представлением обычными «реляционными» средствами. Можно написать запрос, одновременно работающий с реляционными данными, очередями сообщений, XML-данными, пространственными данными, контекстом. И наоборот, создав над реляционными или объектными таблицами базы данных представление XMLType View, можно работать с этими данными через XML-интерфейс.

Поддержка OLAP

Реляционная модель удобна для представления данных в информационно-управляющих системах, однако для аналитических систем более подходит многомерная модель, где данные представлены в виде многомерных кубов, которые можно легко вращать, получать срезы, агрегировать информацию и т. д.

Для создания OLAP-приложений в Oracle ранее использовался программный продукт Express Server — СУБД с многомерной моделью. Данные из оперативных реляционных систем приходилось перегружать или подкачивать в Express Server, который не обеспечивал такого же уровня надежности, масштабирования, защиты, как реляционный сервер Oracle.

Сервер Oracle 9i поддерживает многомерную модель данных, что позволяет пользователю проектировать многомерные кубы и решать, как они будут храниться в Oracle 9i — в реляционных таблицах или в аналитических пространствах (LOB-поля). Обеспечивается возможность переноса данных из базы Express Server в Oracle 9i. Реализован весь набор функций, ранее присущий Express, причем разработчикам Oracle удалось добиться того, что скорость выполнения этих функций была не ниже, чем в Express Server. Алгоритмы добычи данных (data mining) встроены в сервер Oracle 9i.

Механизм Oracle Streams

В СУБД Oracle существует много различных вариантов передачи данных и сообщений о событиях между разными серверами баз данных:

- в случае репликации захватываются и передаются изменения данных и вызовы удаленных процедур;
- в случае работы с очередями сообщений (Advanced Queuing) передается информация о появлении сообщений и сами сообщения;
- в случае резервирования базы данных передаются и применяются к резервной базе архивированные журнальные файлы или их элементы;
- в случае загрузки данных в хранилища данных или Operating Data Store передаются загружаемые данные.

Создан новый единый унифицированный механизм Oracle Streams, передающий данные и сообщения о событиях и объединяющий перечисленные механизмы.

Oracle Streams состоит из трех элементов: захват событий и данных (capture); складирование их в единый упорядоченный по времени информационный поток в едином формате (stage); транспортировка и применение изменений к целевым базам данных (apply).

Oracle 10*g*

Oracle первой предложила СУБД, предназначенную для корпоративных сетей нового типа - систем распределенных вычислений (Grids).

Oracle 10*g* и Grid вычисления предоставляют предприятиям гибкость для удовлетворения меняющихся потребностей бизнеса, высокое качество услуг при небольших расходах, защиту инвестиций и их быструю окупаемость.

Помимо реализации на корпоративном уровне концепции Grid, новая платформа Oracle 10*g* предлагает 10 важнейших усовершенствований:

- рекордное повышение производительности
- самоуправляемость
- автоматическое управление хранением и доступом к данным (ASM)
- обновление ПО и приложений без остановки работы системы
- новые средства обеспечения высокой готовности
- упрощение установки и управления Oracle Real Application Clusters (RAC)
- быстрый перенос частей базы данных между разными платформами
- сокращение времени восстановления при сбоях с минут до секунд
- поддержка огромных баз данных - до 8 эксабайт (10**18)
- новые инструменты web-разработки HTML DB, развитие языка SQL.

Oracle Database 10*g*

Oracle Database 10*g* предназначена для эффективного развертывания на базе различных типов оборудования, от небольших серверов до мощных симметричных многопроцессорных серверных систем, от отдельных кластеров до корпоративных распределенных вычислительных систем. СУБД предоставляет возможность автоматической настройки и управления, которая делает ее использование простым и экономически выгодным. Ее возможности осуществлять управление всеми данными предприятия - от обычных операций с бизнес-информацией до динамического многомерного анализа данных (OLAP), операций с документами формата XML, управления распределенной/локальной информацией - делает ее идеальным выбором для выполнения приложений, обеспечивающих обработку онлайновых транзакций, интеллектуальный анализ информации, хранение данных и управление информационным наполнением.

Oracle Application Server 10*g*

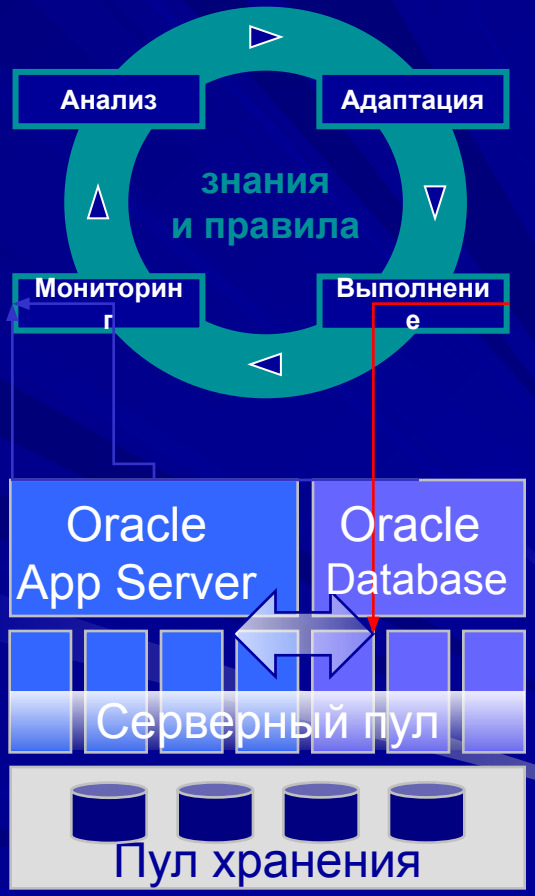
Oracle Application Server 10*g* - это основанная на стандартах интегрированная программная платформа, позволяющая организациям любого масштаба оперативнее реагировать на меняющиеся требования рынка. Oracle Application Server 10*g* обеспечивает полную поддержку технологии J2EE и распределенных вычислений, включает встроенное ПО для корпоративных порталов, высокоскоростного кэширования, интеллектуального анализа бизнес-данных, быстрого развертывания приложений, интеграции бизнес-приложений, поддержки беспроводных технологий, Web-сервисов - и все это в одном продукте. Поскольку платформа Oracle Application Server 10*g* оптимизирована для Grid Computing, она позволяет повысить степень готовности IT-систем и снизить расходы на приобретение аппаратных средств и администрирование.

Oracle Enterprise Manager 10g

Oracle Enterprise Manager 10*g* - это первое в отрасли программное обеспечение, разработанное для администрирования корпоративных сетей распределенных вычислений на базе решений Oracle. Оно призвано помочь снизить сложности, сопряженные с администрированием бизнес-приложений, благодаря управляющему ПО, которое позволяет получить полную информацию обо всей вычислительной инфраструктуре компании. Оно дает системным администраторам возможность реализовать политики, управлять уровнями обслуживания и перераспределять вычислительные ресурсы и приложения при изменении требований бизнеса. Oracle Enterprise Manager 10*g* построено на базе открытой основанной на стандартах архитектуры. Оно поддерживает ключевые стандарты управления, разработанные комитетом Distributed Management Task Force (DMTF), включая Common Information Model (CIM) и Web-based Enterprise Management (WBEM).

Адаптивная платформа для Oracle

- ASCC для Oracle – адаптивная инфраструктурная платформа для приложений и баз данных «по требованию»
- Предложение включает в себя
  - предварительно сконфигурированные серверы, системы хранения и программные компоненты, обеспечивающие автоматизацию, виртуализацию и развёртывание
  - службы для интеграции стандартных и индивидуальных приложений на базе Grid-технологий Oracle Database Server и Oracle Application Server



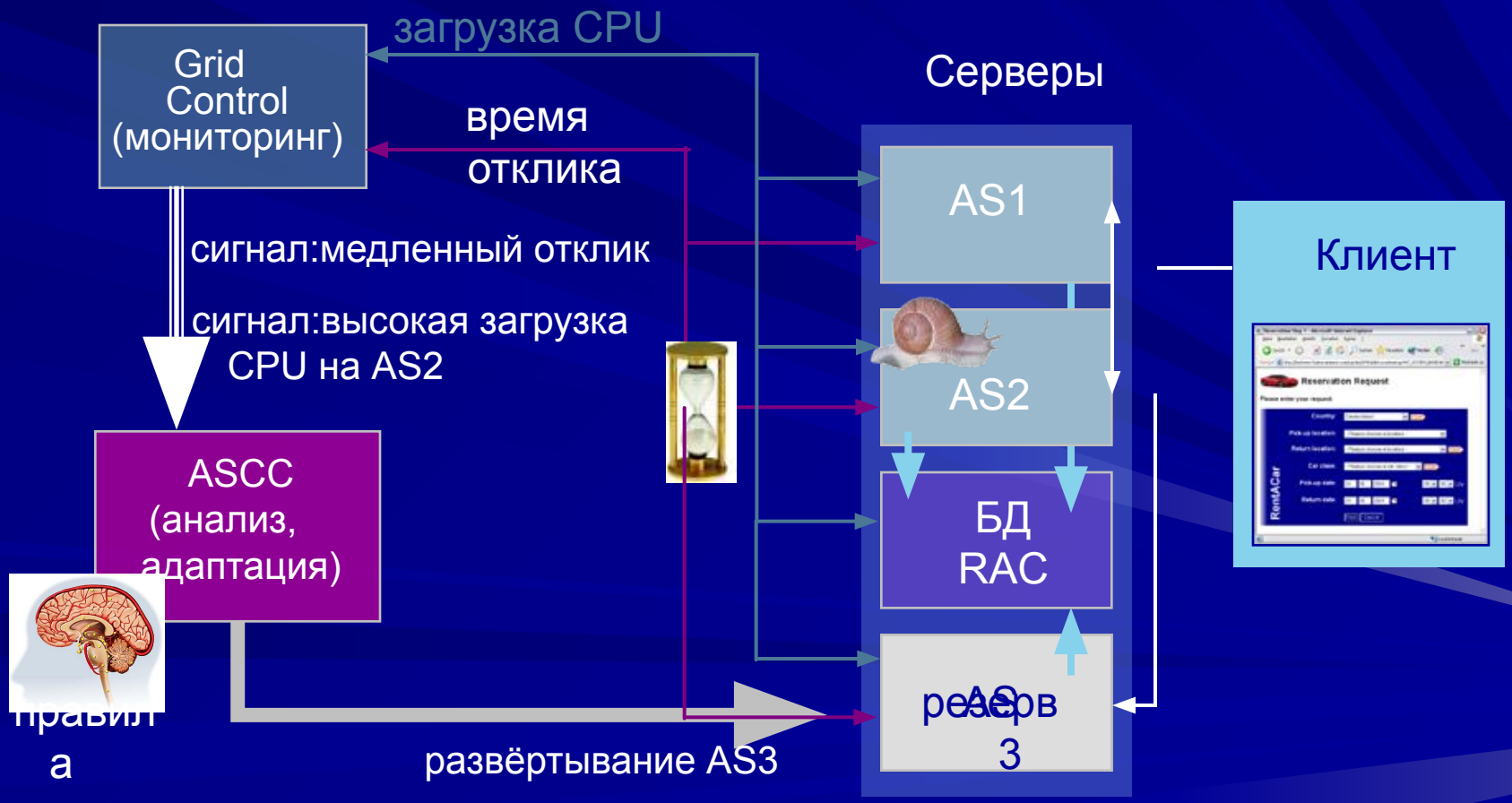
Adaptive Services Control Center (ASCC)

Непрерывный мониторинг физических и виртуальных ресурсов

Ресурсы распределяются адаптивно (в зависимости от загрузки систем)

Для адаптации используются произвольно назначаемые правила и методы

Сценарий: управление загрузкой на базе правил



Сценарий: преодоление сбоя