# СИСТЕМЫ УПРАВЛЕНИЯ

# БАЗАМИ ДАННЫХ

**Часть 2**

## PL/SQL

***Лектор: Иванцова Ольга Владимировна***

# Модель клиент — сервер

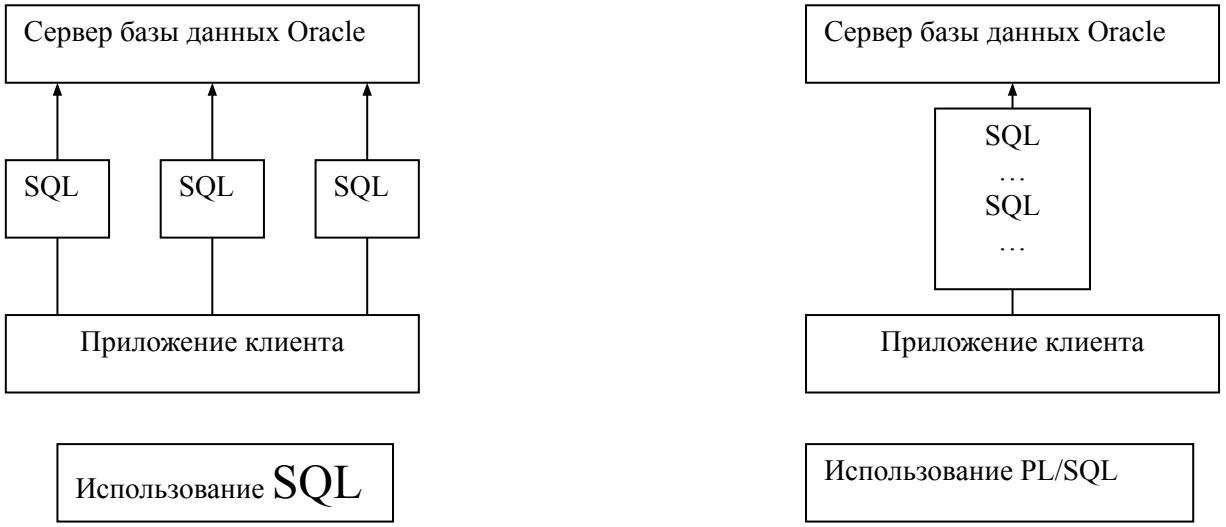



Рис. 1. PL/SQL в среде клиент /сервер

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, как правило, 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.

# Язык PL/SQL

PL/SQL (*Procedural Language extensions to the Structured Query Language*) – процедурные языковые расширения SQL

## ***Характеристики языка PL/SQL***

- Высокоструктурированный
- Стандартизованный и переносимый язык разработки приложений для баз данных Oracle.
- Встроенный язык (функционирует в конкретной хост-среде). Программы на PL/SQL запускаются из БД.
- Высокопроизводительный, высокоинтегрированный язык доступа к БД. Более всего подходит для написания высокоэффективного кода для доступа к БД Oracle.

# Лексические единицы

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.

# Типы данных

- ✔ **Числовые**
- ✔ **Символьные**
- ✔ **Даты и времени**
- ✔ **Логический**
- ✔ **Составные типы: записи и коллекции**
- ✔ **Двоичные типы**
- ✔ **ROWID и UROWID**
- ✔ **REF CURSOR**
- ✔ **Типы данных для поддержки Internet**
- ✔ **ANY**
- ✔ **Объекты (типы данных, определяемые пользователем)**
- ✔ **Предопределенные типы данных объявлены в пакете STANDART.**

# Числовые типы

Основные числовые типы: NUMBER, PLS_INTEGER, BINARY_INTEGER

**NUMBER –** единственный числовой тип, непосредственно поддерживаемый ядром БД.
Number (precision, scale),
где precision – число значащих цифр (от 1 до 38) .
scale – число цифр после запятой (от -84 до 127).

**PLS_INTEGER**

Позволяет хранить целые числа в диапазоне от – 2 147 483 647 до
2 147 483 647. Был разработан для увеличения скорости вычислений.

**BINARY_INTEGER**

Позволяет хранить целые числа со знаком в двоичном формате в диапазоне от – 2 147 483 647 до 2 147 483 647. Не использует встроенную машинную арифметику. Обеспечивает ускорение вычислений при большом объеме операций с целочисленными значениями.

# Числовые подтипы

*Подтипы введены для достижения совместимости с типами ANSI, SQL, SQL/DS, DB2 и представляют собой альтернативные имена для основных типов.*

**NUMBER:**

DEC, DECIMAL, DOUBLE PRECISION, FLOAT, INT, INTEGER, REAL, NUMERIC, SMALLINT

**BINARY_INTEGER:**

NATURAL, NATURALN, POSITIVE, POSITIVEN, SIGNTYPE

# Символьные типы. Набор символов

***Набор символов*** – совокупность символов и соответствующий ей набор битовых последовательностей для представления этих символов в машинном виде.

**ASCII, CP-1251, UNICODE**

**Классифицируется по признакам:**

- ✔ многобайтовый / однобайтовый.
- ✔ фиксированной / переменной длины.

**ASCII –** однобайтовый набор символов фиксированной длины.

**UNICODE UTF-8 –** многобайтовый набор символов переменной длины.

**UNICODE UTF-8 –** многобайтовый набор символов фиксированной длины.

# Набор символов

**С каждой БД ORACLE связаны два набора символов:**

**Набор символов базы данных.**

Используется для представления значений столбцов типа CHAR и VARCHAR2, имен таблиц, столбцов, переменных, строковых литералов.

**Набор символов национального алфавита.**

Используется для представления значений столбцов типа NCHAR и NVARCHAR2, строковых литералов с префиксом N.

*Запрос информации об используемом наборе символов:*

```
SELECT *
FROM nls_database_parameters
WHERE parameter IN ('NLS_CHARACTERSET',
    'NLS_NCHAR_CHARACTERSET');
```

# Строковые типы данных

| Набор символов | Фиксированная длина | Переменная длина |
|---|---|---|
| **Базы данных** | CHAR<br>*Символьные строки фиксированной длины (от 1 до 32 767 байт)* | VARCHAR2<br>*Символьные строки переменной длины (от 1 до 32 767 байт)* |
| **Национальный** | NCHAR<br>*Строки фиксированной длины , состоящие из символов национального алфавита* | NVARCHAR2<br>*Строки переменной длины , состоящие из символов национального алфавита* |

# Строковые подтипы

**VARCHAR2:**

CHAR VARYING, CHARACTER VARYING, STRING, VARCHAR

**CHAR:**

CHARACTER

**NCHAR:**

NATIONAL CHAR, NATIONAL CHARACTER

**NVARCHAR2:**

NATIONAL CHAR VARYING, NATIONAL CHARACTER VARYING, NCHAR VARYING

# Дата и время

**DATE**

Год, месяц, день, часы, минуты, секунды.

**TIMESTAMP**

Дата и время с точностью до миллиардной доли секунды.

**INTERVAL**

Момент, интервал, период.

# Дата и время

Исходный тип данных – DATE.

**Используется для хранения значения даты или даты и времени.**

***Ограничения для типа данных DATE:***

- ✔ **Точность времени – до секунды**
- ✔ **Не содержит информации о часовом поясе**

Тип данных TIMESTAMP(временная метка).

**Используется для хранения времени с точностью до миллиардной доли секунды.**

TIMESTAMP.

**Хранит дату и время без информации о часовом поясе**

TIMESTAMP WITH TIME ZONE.

**Хранит дату и время с информацией о часовом поясе**

TIMESTAMP WITH TIME ZONE.

**Хранит дату и время, соответствующие локальному часовому поясу**

# Дата и время

Типы данных INTERVAL

Момент – **временная точка с некоторой точностью(до часа, до минуты)**

Интервал – **количество времени(час, три часа, пять минут)**

Период – **интервал, который начинается и заканчивается в заданные моменты времени.**

INTERVAL YEAR TO MONTH

-- интервал времени в годах и месяцах

INTERVAL DAY TO SECONDS

-- интервал времени в днях, часах, минутах и секундах(включая доли секунды)

# Тип данных BOOLEAN

Допустимые значения – TRUE, FALSE, NULL.

СУБД ORACLE не поддерживает тип данных BOOLEAN.

Следует учитывать в операторах сравнения, что логическая переменная может принимать значение NULL.

# Составные типы данных

**RECORD (запись)**

похожа на строку из таблицы базы данных, обрабатывается как единое целое.

Не имеет собственного значения. Значение имеет каждый компонент записи.

- ✔ *Запись на основе курсора.*
- ✔ *Запись на основе таблицы.*
- ✔ *Запись, определяемая программистом.*

**TABLE (коллекция) –** составной тип данных, предназначенный для хранения одномерных массивов в программах PL/SQL.

- ✔ *Ассоциативные массивы.*
- ✔ *Вложенные таблицы.*
- ✔ *Массив типа VARRAY.*

# Объявление данных

## *Объявление переменных*

При объявлении переменной ей присваивается имя, задается тип и выделяется память для ее хранения.

имя тип_данных [NOT NULL] [DEFAULT значение_по _умолчанию]|[:= значение_по _умолчанию];

***Примеры:***

```
total  NUMBER;
account  CHAR(15);
userName  VARCHAR2(50);
dateN  DATE  NOT  NULL  DEFAULT  SYSDATE;
```

## *Объявление константы*

имя CONSTANT тип_данных [DEFAULT значение_по _умолчанию]|[:= значение_по _умолчанию];

***Пример:***

```
author CONSTANT VARCHAR2(80) DEFAULT 'Ivanov I.';
NMAX  CONSTANT  PLS_INTEGER := 25;
```

# Объявление с ограничениями

Объявление с указанием ограничений допустимых значений.

*объявление без ограничений:*

для хранения переменной выделяется 38 разрядов

no_limits NUMBER;

*объявление с ограничениями:*

требуется меньше памяти

small NUMBER(1);

large NUMBER(25,6);

title VARCHAR(200);

# Объявления с привязкой

Устанавливается тип данных на основе типа уже определенной структуры данных.

Виды привязки:

- ✔ *Скалярная привязка*. С помощью атрибута %TYPE переменная определяется на основе типа столбца таблицы базы данных или другой скалярной переменной
- ✔ *Привязка к записи*. Через атрибут %ROWTYPE определяется переменная на основе таблицы базы данных или предопределенного явного курсора.

Синтаксис:

```
имя_переменной_ тип_атрибута %TYPE [DEFAULT];
имя_переменной_ имя_таблицы | имя_курсора%ROWTYPE
   [DEFAULT];
```

где

`тип_атрибута` – имя ранее объявленной переменной или спецификация столбца таблицы в формате `таблица.столбец`

# Обработка исключений

*Системное исключение*.

Инициируется исполняемым ядром PL/SQL(NO_DATA_FOUND, TOO_MANY_ROWS, INVALID_NUMBER).

*Исключение, определяемое программистом*.

Определяется в коде PL/SQL, специфично для данного приложения.

Имя исключения связывается с конкретной ошибкой Oracle с помощью директивы компилятора EXCEPTION_INIT.

Присвоить номер исключению и создать для него описание можно с помощью процедуры RAISE_APPLICATION_ERROR.

# Обработка исключений

*Инициировать исключение*.

Остановить выполнение текущего блока PL/SQL путем уведомления исполняемого ядра об ошибке.

*Обработать исключение*.

Перехватить ошибку, передав управление обработчику исключений.

*Неименованное (анонимное) исключение*.

Исключение, с которым связан номер ошибки и описание.

Не имеет имени, поэтому не может быть использовано в операторе RAISE или предложении WHEN обработчика исключений.

*Именованное исключение*.

Исключение, которому присвоено имя.

# Обработка исключений

## Раздел обработки исключений

EXCEPTION
  WHEN *имя_искл_1*
  THEN
  *операторы_обработчика_искл_1;*
  . . .
  WHEN *имя_искл_N* THEN
  *операторы_обработчика_искл_N;*
  END;

## Объявление именованных исключений

*имя_искл_1* EXCEPTION;

# Обработка исключений

*Связывание имени исключения с кодом ошибки*

Коды ошибок – от -20999 до -20000.

SQLCODE – функция, возвращающая код последней сгенерированной ошибки.

Директива EXEPTION_INIT позволяет связать имя объявленной ошибки с некоторым кодом.

```
DECLARE
    имя_исключения EXCEPTION;
    PRAGMA EXEPTION_INIT(имя_исключения, целое_число);
```

Ключевое слово `PRAGMA` указывает, что часть оператора после нее является директивой компилятора. Не включается в исполняемый код.

# Обработка исключений

## Инициирование исключений

*Оператор RAISE*

RAISE имя_исключения;

инициирование системных и объявленных в текущем блоке исключений

RAISE имя_пакета.имя_исключения;

инициирование исключения, объявленного в пакете

RAISE;

повторное инициирование исключения в обработчике исключения

*Процедура RAISE_APPLICATION_ERROR*

Инициирование специфических для приложения исключений. Позволяет связать с исключением сообщение об ошибке.

RAISE_APPLICATION_ERROR (ERRNUM, ERRMES)

ERRNUM – номер ошибки от -20000 до -20999

# 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;**

# Виды блоков

**Допустимы следующие виды блоков:**

- ✔ *Анонимные (непоименованный) 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.*

# Блок PL/SQL

Блоки могут быть вложены друг в друга.

Самый "верхний" блок PL/SQL называется *базовым* и должен заканчивается символом "/".

Этот символ сообщает серверу, что можно приступать к компиляции введенного кода.

Блок, не имеющий заголовка, называется *анонимным*.

*Вложенными* могут быть только анонимные блоки. Они используются в функциях, процедурах и триггерах.

Анонимный базовый блок не сохраняется на сервере, а выполняется сразу.

Если же базовый блок имеет заголовок, то он хранится на сервере в виде скомпилированной процедуры, функции, пакета или триггера( в зависимости от типа заголовка).

# Управляющие структуры PL/SQL

## Оператор CASE

- ✔ Простой.
- ✔ Поисковый.

***Простой***

Связывает одну или несколько последовательностей операторов с соответствующим значением.

CASE *выражение*
WHEN *результат1* THEN
*Операторы1*
WHEN *результат1* THEN
*Операторы1*
…
ELSE
*ОператорыELSE*
END CASE;

# Управляющие структуры PL/SQL

## Оператор CASE

***Поисковый***

Выбирает для выполнения одну из последовательностей операторов в зависимости от результатов вычисления списка логических условий.

CASE
WHEN *выражение1* THEN
*Операторы1*
WHEN *выражение2* THEN
*Операторы2*
…
ELSE
*ОператорыELSE*
END CASE;

# Управляющие структуры PL/SQL

## Оператор GOTO

*Оператор безусловного перехода*

GOTO имя_метки;

...

<<имя_метки>>

Операторы_после_метки;

- ✔ За меткой должен следовать хотя бы один исполняемый оператор.
- ✔ Метка должна находиться в пределах области действия оператора GOTO (функция, процедура, анонимный блок, оператор IF, оператор LOOP, обработчик исключения, оператор CASE).
- ✔ Метка должна находиться в той же части блока, что и оператор GOTO.

## Оператор NULL

Используется для указания компилятору “ничего не делать”: NULL.

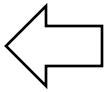
# Управляющие структуры PL/SQL

Структуры управления являются основой любого языка программирования, т. к. 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.



# Условный оператор IF

**Три типа условного оператора if**

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
        P1    NUMBER;
              P2    NUMBER;
        P_REZ    VARCHAR2(7);
            BEGIN
               …..
                    IF     P1< P2       THEN
                                P_REZ := ‘YES’;
                    ELSE
                                P_REZ := ‘NO’;
                   END   IF;
            END;
```

# Циклы

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

***Четыре вида операторов цикла***.

## 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

# 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;
```

# 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.

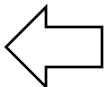
Курсоры представляют собой области памяти, специально предназначенные для обработки этих операторов. В одних случаях курсоры *объявляются явно*, а других программист предоставляет *PL/SQL самому выполнить эту операцию*.

**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 ) 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,

а *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:

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,

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

*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 */
            N_z     Студенты. N_зачетки %TYPE;
            F_Name  Студенты. Имя%TYPE;
            L_Name  Студенты. Фамилия%TYPE;

        /*  Переменная привязки, используемая в запросе*/
            N_gr    Студенты. Группа%TYPE := 1011;

        /*  Создание курсора*/
        CURSOR c_Students IS
            SELECT N_зачетки, Имя, Фамилия
            FROM Студенты
               WHERE   Группа = N_gr;
BEGIN
        /*Обозначим строки активного набора*/
           OPEN c_Students
          LOOP
       /*Выберем каждую строку  активного набора в переменные PL/SQL*/
            FETCH c_Students  INTO   N_z, F_Name , L_Name ;
       /*  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  Хобби
                SET Риск= 10
                 WHERE  Название_Хобби=  ‘Бокс’;
/*   Если предыдущий оператор  UPDATE не выбирает ни одной строки, то введем новую строку в таблицу
      Хобби*/
       IF     SQL%NOTFOUND THEN
              INSERT      INTO    Хобби     VALUES (‘Бокс’,100);
        END IF;
END;
```

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

```
  BEGIN
UPDATE  Хобби
                SET Риск= 10
                 WHERE  Название_Хобби=  ‘Бокс’;
/*   Если предыдущий оператор  UPDATE не выбирает ни одной строки, то введем новую строку в таблицу
      rooms*/
                IF     SQL%ROWCOUNT     THEN
              INSERT      INTO  Хобби     VALUES (‘Бокс’,100);
                END IF;
END;
```

# Процедуры

**Создание процедуры**

Синтаксис оператора CREATE OR REPLACE PROCEDURE таков:

**CREATE [OR REPLACE] PROCEDURE *имя_процедуры***

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

**. . .**

**а*ргумент* [{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.

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.

Если процедура существует, она сначала удаляется безо всякого предупреждения, для чего используется команда **DROP PROCEDURE**. Если процедура до этого не

# 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*];**

## 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.

# Функции

**Создание функций**

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

**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.

# Функции и процедуры

Как и для процедур, *список аргументов функции 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.

*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.*

**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.

**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*.**

# 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.

# 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;
```

# Модули

**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.

# Правила для заголовка модуля

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;


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;
END  Random;
```

```
BEGIN
/* Инициализация модуля.  Инициализируем  исходное  значение
     текущим временем  в секундах */
 Random.ChangeSeed(TO_NUMBER (TO_CHAR(SYSDATE, 'SSSS')));

END;
/
```

*Для получения случайного числа можно просто вызвать*

```
SELECT  Random.Rand   from  DUAL;
```

**Последовательность случайных чисел зависит от исходного значения – для одного и того же исходного значения генерируются одинаковые последовательности.**

# 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.

*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 срабатыватие 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.

# Создание триггеров

**CREATE [OR REPLACE] TRIGGER** *имя_триггера*
**{BEFOR | 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 т.п.

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

# Элементы триггера и имена триггеров

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

*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 они 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;
```

**ПРИМЕРЫ ТРИГГЕРОВ**

```
CREATE  [OR  REPLACE]  TRIGGER  classesBEstatement
    BEFOR  UPDATE  ON  classes
BEGIN
    INSERT   INTO   temp_table (num_col, char_col)
              VALUES (trigger_seq.NEXTVAL, 'BEFOR 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
   BEFOR UPDATE ON classes
   FOR EACH ROW
BEGIN
   INSERT INTO temp_table (num_col, char_col)
         VALUES (trigger_seq.NEXTVAL, 'BEFOR 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 BEFOR,

:old никогда не модифицируется, а лишь считывается.

# Использование :old и :new в строковых триггерах

| **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** BEFOR, срабатывающего на операторы INSERT и UPDATE, заполняющий поле N_порядковый в таблице Студенты_Хобби значением, генерируемым последлвательностью student_seq.

# Примеры триггеров

1. *Триггер BEFOR, срабатывающий на операторы INSERT и UPDATE, заполняющий поле ID в таблице Студенты_Хобби значением, генерируемым последлвательностью student_seq.*

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

2. *Триггер BEFOR, запрещающий редактировать данные в таблице Студенты в период с 18 часов до 9 часов.*

```
CREATE TRIGGER UP_DEL_Time
     BEFORE UPDATE OR DELETE ON Студенты
    BEGIN
      IF to_char (sysdate, 'HH24') NOT BETWEEN 9 or 18 THEN
RAISE_APPLICATION_ERROR (-20101, 'НЕЛЬЗЯ ИЗМЕНЯТЬ ДАННЫЕ О СТУДЕНТАХ В ЭТО ВРЕМЯ');
          ENDIF;
      END UP_DEL_Time ;
```

# Триггеры INSTEAD OF(замещающие триггеры)

Предназначены для выполнения операций вставки, обновления и удаления элементов ***представлений***.

*Создание триггера INSTEAD OF*

**CREATE [OR REPLACE TRIGGER]** ***имя_триггера***
**INSTEAD OF** ***операция*** **ON** ***имя_представления***
**[DECLARE]**
**BEGIN**
**…**
**END;**

**Пример**

Система учета доставки базируется на трех таблицах:

- ✔ delivery(учет доставленной продукции),
- ✔ area(список районов доставки),
- ✔ driver(учет объемов работы курьеров).

TABLE delivery

| COLUMN_NAME | COLUMN_TYPE |
|---|---|
| delivery_id | NUMBER |
| delivery_start | DATE |
| delivery_end | DATE |
| area_id | NUMBER |
| driver_id | NUMBER |

TABLE area

| COLUMN_NAME | COLUMN_TYPE |
|---|---|
| area_id | NUMBER |
| area_desc | VARCHAR2(30) |

TABLE driver

| COLUMN_NAME | COLUMN_TYPE |
|---|---|
| driver_id | NUMBER |
| driver_name | VARCHAR2(30) |

***Объединяем всю информацию в одно представление:***

```
CREATE OR REPLACE VIEW delivery_info AS
SELECT d.delivery_id, d.delivery_start, d.delivery.end, a.area_desc,
dr.driver_name
FROM delivey d, area a, driver dr
WHERE a.area_id = d.area_id AND dr.driver_id = d.driver_id;
```

***Триггер INSTEAD OF INSERT должен:***

✔ Гарантировать, что столбец delivery.end будет содержать NULL.

✔ Определить идентификатор курьера по имени. Если указанное имя в таблице отсутствует, триггер присвоит курьеру новый идентификатор и добавит строку в таблицу курьеров.

✔ Определить идентификатор района по названию. Если указанное название в таблице отсутствует, триггер присвоит району новый идентификатор и добавит строку в таблицу районов.

*курсор для получения идентификатора курьера по его имени:*

```
CURSOR cur_get_driver_id(cp_driver_name VARCHAR2) IS
SELECT driver_id FROM driver WHERE driver_name = cp_driver_name;
v_driver_id NUMBER;
```

*курсор для получения идентификатора района по его названию:*

```
CURSOR cur_get_area_id(cp_area_desc VARCHAR2) IS
SELECT area_id FROM area WHERE area_desc = cp_area_desc;
v_area_id NUMBER;
```

*значение столбца delivery_end должно быть NULL*

```
IF :new.delivery_end IS NOT NULL THEN
    RAISE_APPLICATION_ERROR(-20000, 'Delivery end date value must be
     NULL when delivery created');
END IF;
```

*/* Получить идентификатор курьера по его имени. Если имя не найдено, создать новый идентификатор на последовательности. Вставить строку в таблицу. */*

```
OPEN cur_get_driver_id(:new.driver_name);
FETCH cur_get_driver_id INTO v_driver_id;
IF cur_get_driver_id%NOTFOUND THEN
SELECT driver_id_seq.NEXTVAL INTO v_driver_id FROM DUAL;
INSERT  INTO driver(driver_id,driver_name) VALUES(v_driver_id,:new.driver_name);
END IF;
CLOSE cur_get_driver_id;
```

*/* Получить идентификатор района по его названию. Если название не найдено, создать новый идентификатор на последовательности.*/*

```
OPEN cur_get_area_id(:new. area_desc);
FETCH cur_get_area_id INTO v_area_id;
IF cur_get_area_id%NOTFOUND THEN
SELECT area_id_seq.NEXTVAL INTO v_area_id FROM DUAL;
INSERT  INTO area(area_id, area_desc) VALUES(v_area_id,:new.area_desc);
END IF;
CLOSE cur_get_area_id;
```

**Триггер INSTEAD OF UPDATE** *обновляет столбец delivery_end, если он содержит NULL*

```
CREATE OR REPLACE TRIGGER delvery_info_update
INSTEAD OF UPDATE ON delivery_info
DECLARE
--курсор для получения строки доставки
CURSOR cur_get_delivery(cp_delivery_id NUMBER) IS
      SELECT delivery_end FROM delivery
       WHERE delivery_id = cp_delivery_id FOR UPDATE OF delivery_end;
            v_delivery_end DATE;
       BEGIN
OPEN cur_get_delivery(:NEW.delivery_id)
FETCH cur_get_delivery INTO v_delivery_end;
        IF v_delivery_end IS NOT NULL THEN
    RAISE_APPLICATION_ERROR(-20000,'The delivery end date has already been set');
        ELSE
  UPDATE delivery SET delivery_end = :NEW. delivery_end WHERE CURRENT OF
     cur_get_delivery;
END IF;
CLOSE cur_get_delivery;
END;
```

# Триггер INSTEAD OF DELETE

*Следить, чтобы заполненные строки не удалялись. А затем самому удалить указанную строку доставки.*

```
CREATE OR REPLACE TRIGGER delivery_info_delete
INSTEAD OF DELETE ON delivery_info
BEGIN
  IF :OLD.delivery_end IS NOT NULL THEN
     RAISE_APPLICATION_ERROR(-20000,'Completed deliveries
    cannot be deleted');
END IF;
DELETE delivery WHERE delivery_id = :OLD.delivery_id;
END;
```

# 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.**

# Системы объектно-ориентированных баз данных

**Обязательные свойства**

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 критрий* 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.

# 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.

СУБД Oracle, начиная с 8 версии 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.

# Объектно-реляционные базы данных

В объектно-реляционной базе данных язык SQL (и PL/SQL) используется для манипулирования как реляционными, так и объектными данными.

Кроме того, СУБД Oracle 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.

# Достоинства и недостатки реляционной и объектной моделей

***Задачи, недоступные для реляционного подхода***

Реляционной моделью трудно описать предметную область, подразумевающую наличие разветвленной иерархии множества объектов или большого числа сложных типов данных.

Задача еще более осложняется, если в ходе эксплуатации ИС постоянно требуется вводить новые типы данных.

Если кратко, то реляционные базы данных не позволяют использовать всю полноту и функциональность объектно-ориентированного подхода, который стал очень популярным в последнее время.

***Задачи, недоступные для объектного подхода***

Внутри объектные базы данных построены не на реляционной алгебре, хотя и имеют SQL-интерфейс. Реляционные же базы данных позволяют быстрее выполнять SQL-запросы, поскольку используют реляционную алгебру на уровне ядра.

# Задачи, недоступные обеим моделям

Пусть мы собираем информацию о лицах в течение длительного периода времени.

База проектировалась 10 лет назад, и в ней имелся атрибут личности *«партийность»*.

Для него был определен вполне достаточный логический тип.

Через несколько лет вдруг выяснилось, что логического типа уже недостаточно, и для «партийности» требуется указывать значение из списка.

Еще через несколько лет графа «партийность» в организации, ведущей базу данных, оказалась упразднена. Таким образом, для данных об одном и том же лице, относящихся к одному периоду времени, должен использоваться булев атрибут «партийности», для данных, относящихся к другому периоду, – символьный атрибут (внешний ключ?), а для данных, относящихся к третьему периоду, атрибут должен отсутствовать. Другими примерами структурных изменений во времени могут служить упразднение атрибута «национальность» или добавление атрибута «государственный пенсионный страховой номер».

Сегодняшние подходы к моделированию данных (что реляционный, что объектный) сориентированы в первую очередь на статическое моделирование прикладной области и мало рассчитаны на динамику (историю).

В лучшем случае можно долго и тщательно разрабатывать схему данных, а уж потом фиксировать ее и реализовывать в ИС, но никогда не эксплуатируется тезис о том, что «структура данных может изменяться (а не только пополняться)».

# Реализация объектного подхода в ORACLE

В целом объектная реализация в ORACLE традиционна для объектного подхода вообще.

Строго говоря, СУБД ORACLE не является объектной.

Точнее, в СУБД ORACLE есть *объектное расширение* – возможности по созданию и использованию объектов, причем возможности эти несколько ограничены по сравнению с большинством языков высокого уровня.

# Хранение объектов в столбцах реляционных таблиц

***Например, для описания некоторого абстрактного адреса создадим тип*:**

```
CREATE OR REPLACE TYPE address_typ AS OBJECT(
Zip   CHAR(6),
City   VARCHAR2 (20),
Location VARCHAR2(50)   ) ;
      /
```

***воспользуемся этим типом для описания отделов*:**

```
CREATE   TABLE   DEPARTMENTS(
Dep_Id   NUMBER   CONSTRAINT   departments_pk   PRIMARY KEY,
Caption VARCHAR2 (50),
Addr address_typ);

CREATE   SEQUENCE   Departments_seq;

INSERT   INTO   Departments   VALUES
(Departments_seq.NEXTVAL, 'Отдел продаж',
 NEW address_typ('141980', 'Дубна', 'Университетская, 19'));
```

***выполним запросы к полям, содержащим объекты*:**

```
SELECT  d.Caption,  d.Addr.ZIP,  d.Addr.City,  d.Addr.Location   FROM   Departments d;
```

# Создание таблицы объектов

Кроме хранения объектов в полях реляционных таблиц существует возможность хранить объекты и в специальных объектных таблицах – таблицах, хранящих только объекты одного какого-то типа.

*Пример создания объектной таблицы для хранения адресов:*

**CREATE TABLE Addresses OF address_typ;**

Таблицы объектов в ORACLE было бы точнее называть списками объектов, так как это всегда таблицы ровно из одного столбца объектного типа.

*заполнение данными происходит, как и ранее:*

**INSERT INTO Addresses VALUES(**
**NEW address_typ('141980', 'Дубна', 'Университетская, 19'));**

*либо с неявным вызовом конструктора по умолчанию*

**INSERT INTO Addresses VALUES('141980', 'Дубна', 'Университетская, 19');**

*для выборки отдельных свойств объектов из объектных таблиц*

**SELECT a.zip, a.city, a.location FROM Addresses a;**

*для выборки всего объекта целиком (например, для передачи в качестве параметра какой-нибудь процедуре) используется функция VALUE:*

**SELECT VALUE(a) FROM Addresses a;**

# Ссылки на объект

*Объекты, хранимые в объектных таблицах, имеют одно преимущество перед объектами, хранимыми как атрибут строки: на них можно ссылаться.*

Ссылка - есть уникальный внутренний идентификатор объекта (адрес объекта в базе данных), и получить его можно с помощью функции *REF*:

**SELECT REF(a) FROM Addresses a;**

В случае объектного подхода, задача контроля ссылочной целостности полностью лежит на программисте.

**Примечание**: ссылка представляет собой уникальный идентификатор таблицы, в которой содержится объект плюс уникальный идентификатор супертипа, от которого унаследован объект и, наконец, уникальный идентификатор строки таблицы (ROWID), на который ссылается объект.

В итоге получается длинное число, которое можно использовать при сравнении с другими ссылками (например, для определения, ссылаются ли ссылки на объекты одного и тоже супертипа).

Для демонстрации работы со ссылками на объект несколько изменим структуру таблицы DEPARTMENTS:

ALTER TABLE DEPARTMENTS DROP (Addr);

ALTER TABLE DEPARTMENTS ADD (Addr REF address_typ SCOPE IS Addresses);

Фраза *SCOPE IS* при определении типа как ссылки на существующий объект необязательна, но позволяет фактически ссылаться только на объекты какой-то определенной объектной таблицы.

Если параметр SCOPE IS не указан, то подразумевается, что поле Addr в таблице Departments может ссылаться на объекты типа address_typ, которые могут находиться в какой угодно таблице. Другими словами, разные записи одного поля таблицы могут ссылаться на объекты, физически расположенные в разных таблицах.

**UPDATE Departments d**

**SET d.Addr=(SELECT REF(a) FROM Addresses a**

**WHERE VALUE(a)= address_typ('141980', 'Дубна', 'Университетская, 19'));**

*Ссылочный тип отличается от реляционного FOREIGN KEY ограничения тем, что он обеспечивает выборку данных по значению, на которое указывает ссылка (не нужно внутреннее объединение таблиц внутри одного запроса):*

**Select d.Caption, DEREF(d.Addr).ZIP, DEREF(d.Addr).City, DEREF(d.Addr).Location From Departments d;**

**Примечание**: функция *DEREF* применяется для явного раскрытия ссылки, то есть для получения объекта, который находится по указанному адресу. При этом СУБД самостоятельно определяет таблицу, из которой необходимо достать объект.

*В отличие от внешнего ключа, поле-ссылка может указывать на несуществующий объект. Возникает понятие повисшей (не существующей ссылки).*

*Этот факт можно выявить с помощью оператора DANGLING («повисший»):*

-- удалим строку из таблицы Addresses;

**Delete from Addresses;**

-- выберем строки, содержащие повисшие ссылки

**SELECT d.Caption FROM Departments d Where Addr IS DANGLING;**

-- выберем строки, содержащие действительные ссылки

**SELECT d.Caption FROM Departments d Where Addr IS NOT DANGLING;**

# Конструкторы

Каждый создаваемый объект автоматически имеет конструктор по умолчанию.

Об этом заботится сама СУБД.

Но программист может сам создать альтернативный конструктор, использую параметр CONSTRUCTOR в объявлении метода объекта.

Для того чтобы метод являлся конструктором, он должен удовлетворять ряду ограничений:

- ✔ название метода должно совпадать с названием объекта;
- ✔ метод должен являться функцией, возвращающей тип SELF в качестве результата;
- ✔ количество и типы формальных параметров функции не должны полностью совпадать с количеством и типом свойств объекта.

```
create or replace type some_typ IS OBJECT(
        name varchar2(30),
    CONSTRUCTOR FUNCTION some_typ RETURN SELF AS RESULT,
    CONSTRUCTOR FUNCTION some_typ(N NUMBER)  RETURN  SELF  AS RESULT);
/
 create or replace type body some_typ IS
        CONSTRUCTOR FUNCTION some_typ RETURN  SELF  AS  RESULT   IS
    BEGIN
            Name:='Unknown';
            RETURN;
    END;
        CONSTRUCTOR FUNCTION some_typ(N  NUMBER) RETURN SELF AS
   RESULT  IS
    BEGIN           Name:=TO_CHAR(N);
            RETURN;
    END;
END;
/
```

```
declare
        s1  some_typ;          s2  some_typ;          s3  some_typ;
begin
вызов конструктора по умолчанию
    s1:=some_typ('Pete');
        dbms_output.put_line(s1.Name);
 вызов первого конструктора без параметров
        s2:=some_typ();
    dbms_output.put_line(s2.Name);
 вызов второго конструктора с одним параметром
    s3:=some_typ(10);
        dbms_output.put_line(s3.Name);
end;
/
```

*Результат:*

Pete

Unknown

10

# Статические методы

Методы класса являются статическими в том случае, если в их объявлении был указан параметр STATIC.

Такие методы, по аналогии со статическими методами в других объектно-ориентированных языках программирования, могут вызываться без создания экземпляра объекта.

```
CREATE OR REPLACE TYPE datetime IS OBJECT(
        CurDateTime DATE,
    STATIC PROCEDURE PrintDateTime);
/
 CREATE OR REPLACE TYPE BODY datetime IS
        STATIC PROCEDURE PrintDateTime AS
    BEGIN
        Dbms_output.put_line(sysdate);
    END;
END;
/
```

```
set serveroutput on
begin
    datetime.PrintDateTime;
end;
/
```

**Примечание**: нельзя создать объект, состоящий только из статических методов.

При необходимости объединения внутри одного объекта нескольких статических методов, используют пакеты.

# Методы сравнения

При выполнении операторов ORDER BY, GROUP BY, DISTINCT или операторов сравнения СУБД должна уметь сравнивать объекты между собой.

Методы сравнения объектов между собой, используемые СУБД по умолчанию, не всегда могут удовлетворять решаемым задачам, поэтому есть возможность переопределить методы сравнения (аналог перегрузки операции сравнения).

При описании метода объекта можно воспользоваться одним из модификаторов – MAP или ORDER.

***Метод типа MAP*** возвращает значение одного из встроенных типов, которое может использоваться в операциях сравнения.

***Метод типа ORDER*** сравнивает два объекта и возвращает –1, если первый объект меньше, 0, если оба равны, 1, если первый объект больше.

Сравнение объектов, использующих метод сравнения типа MAP, гораздо быстрее, чем объектов, использующих метод сравнения типа ORDER.

Если метод сравнения не задан вообще, объекты этого типа можно сравнивать только на равенство/неравенство. При этом объекты будут считаться равными, только если все их атрибуты не-NULL и равны.

```
CREATE OR REPLACE TYPE datetime IS OBJECT(
        CurDateTime DATE,
    STATIC PROCEDURE PrintDateTime,
        ORDER MEMBER FUNCTION compare(other datetime) RETURN NUMBER);
/

CREATE OR REPLACE TYPE BODY datetime IS
   STATIC PROCEDURE PrintDateTime AS
   BEGIN
        Dbms_output.put_line(sysdate);
    END;
     ORDER MEMBER FUNCTION compare(other datetime) RETURN NUMBER IS
        BEGIN
            IF  SELF.CurDateTime<other.CurDateTime THEN  RETURN  -1;
              ELSIF SELF.CurDateTime=other.CurDateTime THEN  RETURN  0;
               ELSE  RETURN  1;
             END IF;
         END;
END;
/
```

```
set serveroutput on
declare
        dt1 datetime:=datetime(sysdate);
         dt2 datetime:=datetime(sysdate+1);
begin
        if   dt1<dt2    then
        dbms_output.put_line(dt1.CurDateTime || ‘ меньше, чем ’
                                                || dt2.CurDateTime);
         else
        dbms_output.put_line(dt1.CurDateTime || ‘ больше или равно, чем ’
                                                || dt2.CurDateTime);
end if;
end;
/
```

30-NOV-10 меньше, чем 01-DEC-10

# Объектные представления

В силу разных обстоятельств может оказаться удобной имитация объектов на основе данных, хранимых в традиционных таблицах.

Тогда к одним и тем же данным можно обращаться и через объектный интерфейс, и через реляционный.

Достигается это с помощью объектных представлений (object views), которые можно так назвать по аналогии с представлениями (views).

**Объектное представление** – *это представление с одной колонкой (очень похожи на таблицы объектов), где хранятся объекты, формирующиеся динамически на основе табличных данных.*

**Примечание**: понятие «обновляемое представление» применимо и к объектным представлениям.

Есть несколько характеристик, которым должны удовлетворять представления для того, чтобы они были обновляемыми (для того чтобы к ним напрямую можно было применять команды DML).

Если же представления не являются обновляемыми, а применять команды DML нужно, используются INSTEAD OF триггеры.

# Особенности наследования объектов в ORACLE

Реализация механизма наследования в СУБД ORACLE имеет свои особенности по сравнению с такими языками высокого уровня как, например, Delphi (Object Pascal) или C++.

Отличия, в частности, заключаются в следующем:

- ✔ нет корневого типа, от которого наследуются все остальные объекты (как TObject в Delphi).
- ✔ Нет множественного наследования, только простое (наследование от одного предка).
- ✔ Терминология механизма наследования, реализованного в СУБД ORACLE, также несколько отличается от более привычной терминологии для тех, кто программирует на языках высокого уровня (см. таблицу).

| **Термин в ООП** | **Термин в СУБД ORACLE** |
|---|---|
| Класс | Объект |
| Экземпляр класса | Экземпляр объекта |
| Родительский класс | Супертип |
| Класс-потомок | Подтип |

Для того чтобы указать, что новый объект должен унаследовать все свойства и методы супертипа (родительского объекта), используют следующую конструкцию:

**CREATE OR REPLACE TYPE имя_подтипа UNDER имя_супертипа( …);**

*По умолчанию типы в ORACLE не могут быть унаследованы.*

Для того чтобы тип мог быть в последствии унаследован другим, в объявлении этого типа должен быть указан параметр *NOT FINAL* (такие типы называются *нетерминальными*):

```
CREATE OR REPLACE TYPE имя_типа IS OBJECT(
    …
)
NOT FINAL;
```

# Абстрактные объекты

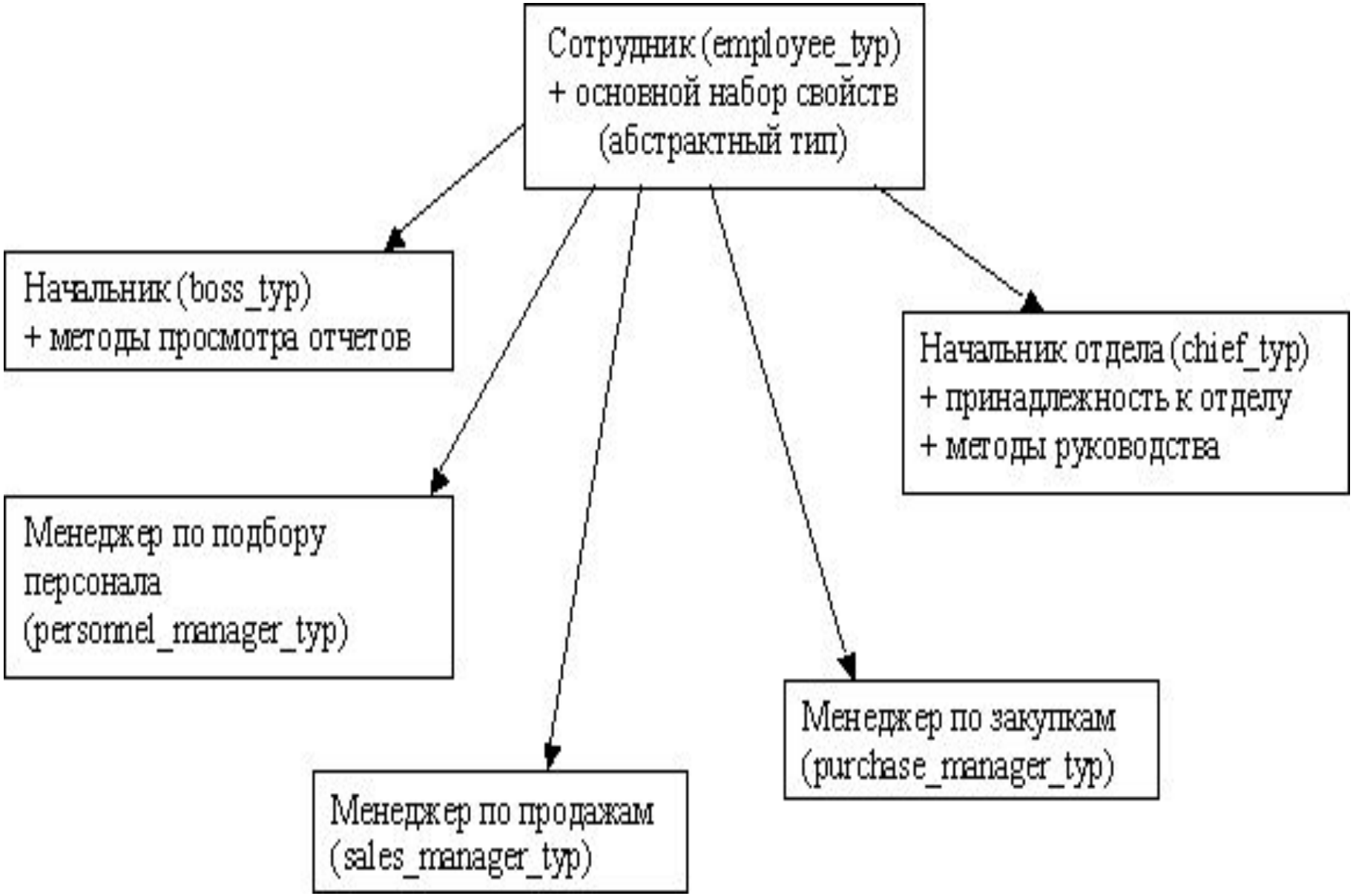
***Абстрактные объекты*** – это объекты, экземпляры которых нельзя создать (по аналогии с абстрактными классами в ООП).

Такие объекты создаются для того, чтобы в последствии выступить в качестве супертипов в наследовании.

Для того чтобы сделать объект абстрактным, в его объявление надо добавить параметр *NOT INSTANTIABLE*.

Указание параметра *NOT FINAL* в этом случае является обычным, так как мало практического смысла в том, чтобы создавать абстрактный объект, который нельзя унаследовать.

# Пример иерархии объектов



*-- создадим объект department_typ11 для хранения информации об отделе:*

```
CREATE OR REPLACE TYPE department_typ11 IS OBJECT(Dep_Id
    NUMBER,
                                                    Caption
    varchar2(50));
/
```

*-- создадим абстрактный объект employee_typ11, обладающий основным набором свойств, а также метод, определяющий количество отработанных дней:*

```
CREATE OR REPLACE TYPE employee_typ11 IS OBJECT(
     Name VARCHAR2(30),
     Hire_Day DATE,
  MEMBER FUNCTION DaysAtCompany RETURN NUMBER)
 NOT INSTANTIABLE
 NOT FINAL;
/
 CREATE OR REPLACE TYPE BODY employee_typ11 IS
MEMBER FUNCTION DaysAtCompany RETURN NUMBER IS
        BEGIN
            RETURN TRUNC(sysdate-Hire_Day);
         END;
 END;
/
```

*-- создадим объект boss_typ, наследующий все свойства и методы объекта employee_typ11 (для простоты примера данный объект не будет иметь собственных свойств и методов)*

**CREATE OR REPLACE TYPE boss_typ11 UNDER employee_typ11();**

**/**

*-- создадим объект sales_manager_typ11, наследующий все свойства и методы объекта employee_typ11 и имеющий дополнительное свойство, обозначающее принадлежность к отделу*

**CREATE OR REPLACE TYPE sales_manager_typ11 UNDER employee_typ11(**

**Dep department_typ11);**

**/**

***Примечание:*** данные о взаимозависимостях типов можно посмотреть в таблице **USER_TYPES:**

```
SELECT supertype_name, type_name
FROM  user_types
ORDER BY 1, 2;
```

# Реализация полиморфизма в СУБД ORACLE

Под полиморфизмом в ORACLE понимается возможность использования *свойства подстановочности* (substitutability) и *динамической диспетчеризации методов* (Dynamic Method Dispatch), иными словами, виртуальных методов.

## Полиморфизм типов

*Подстановочность* – это основная характеристика полиморфизма типов (type polymorphism), которая позволяет использовать значение некоторого подтипа там, где ожидается значение супертипа.

*-- создадим таблицу для хранения информации об отделах и наполним ее данными:*

```
CREATE TABLE Deps OF department_typ11;

    INSERT INTO Deps VALUES(1, 'Отдел продаж');
    INSERT INTO Deps VALUES(2, 'Отдел закупок');
    INSERT INTO Deps VALUES(3, 'Отдел кадров');
```

*-- создадим таблицу для хранения информации о сотрудниках и последовательность для генерации значений первичных ключей таблицы:*

```
CREATE TABLE Employees11( emp_id NUMBER,  employee employee_typ11);
CREATE SEQUENCE Employees_seq;
```

*Рассмотрим, каким образом можно хранить данные о разных сотрудниках в одной таблице:*

*-- добавление босса:*

**INSERT INTO Employees11 VALUES(Employees_seq.NEXTVAL, boss_typ11('Петр Иванович Сидоров', to_date('12.03.2004','dd.mm.yyyy')));**

*-- добавление менеджера по продаже:*

**INSERT INTO Employees11 VALUES(Employees_seq.NEXTVAL, sales_manager_typ11( 'Вася', to_date('12.03.2004','dd.mm.yyyy'), department_typ11(1,'Отдел продаж')));**

*-- выборка перечня сотрудников предприятия:*

SELECT * FROM Employees11;

| EMP_ID | EMPLOYEE(NAME, HIRE_DAY) |
|---|---|
| 1 | BOSS_TYP11('Петр Иванович Си-доров', '12-MAR-04') |
| 2 | BOSS_TYP11('Петр Иванович Си-доров', '12-MAR-04') |
| 3 | SALES_MANAGER_TYP11('Вася', '12-MAR-04', DEPARTMENT_TYP11(1, 'Отдел продаж')) |

# Расширение и сужение объектных типов

Расширение и сужение, очень полезные при работе с объектами в иерархии типов.

***Расширение*** – это присвоение, в котором объявленный *тип источника является более конкретным*, чем объявленный тип места назначения (например, присвоение переменной типа sales_manager_typ значения типа employee_typ).

***Сужение*** – это присвоение, в котором *объявленный тип источника является более общим*, чем объявленный тип места назначения (например, присвоение переменной типа employee_typ значения переменной sales_manager_typ).

Примеры неявного расширения были рассмотрены ранее (присвоение столбцу типа employee_typ значений типов boss_typ и sales_manager_typ). Рассмотрим, как выполняется более сложный шаг – сужение.

# Функция TREAT

ORACLE предоставляет специальную функцию *TREAT*, которая позволяет выполнять операцию сужения, она явно изменяет объявленный тип источника в присваивании на более специализированный тип (подтип) в иерархии места назначения.

Без использования этой функции нельзя сослаться на специфичные для подтипа атрибуты и методы.

*-- общий синтаксис функции:*

**TREAT (<object_instance> AS <object_type>)**

где <object_instance> – это значение столбца или строки коллекции данного конкретного супертипа в объектной иерархии,

<object_type> – это подтип в этой иерархии.

*-- выполним запрос:*

**SELECT e.employee.Name FROM Employees e;**

*-- не удастся выполнить следующий запрос, поскольку свойство Dep есть только у типа sales_manager_typ и его подтипов:*

**SELECT e.employee.Dep.Caption FROM Employees;**

*-- для выполнения предыдущей задачи необходимо выполнить запрос:*

**SELECT TREAT(employee as sales_manager_typ).Dep.Caption FROM Employees;**

***Для объектов отличных от sales_manager_typ функция TREAT вернет значение NULL.***

# Оператор IS OF

Для того чтобы отобрать объекты какой-то конкретной ветки иерархии объектов используют *оператор IS OF*.

Например, следующий запрос вернет имена только тех объектов из поля employee таблицы Employees, которые имеют тип boss_typ или более конкретный:

**SELECT e.employee.Name FROM Employees e WHERE employee IS OF (boss_typ);**

Для того чтобы в выборке присутствовали объекты только одного какого-то типа, оператор IS OF дополняют *параметром ONLY*:

**SELECT e.employee.Name FROM Employees e WHERE employee IS OF (ONLY boss_typ);**

# Параллельные архитектуры серверов баз данных

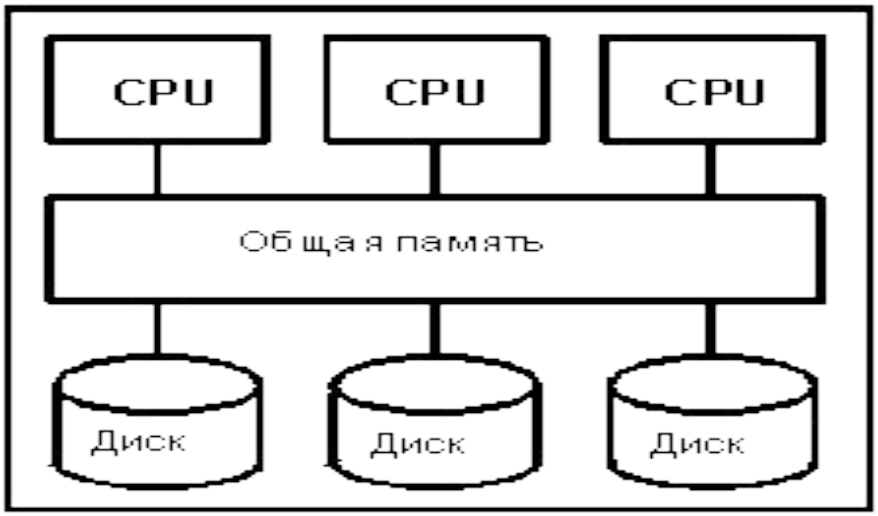
***Основные архитектурные направления***:

- ✔ Симметричные многопроцессорные системы (SMP) - форма сильносвязанных многопроцессорных систем, разделяющих единую оперативную память и дисковую подсистему;

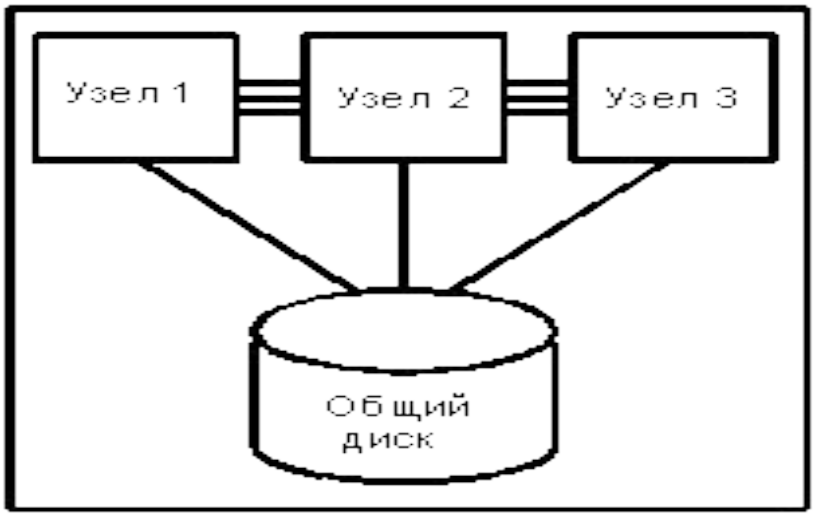
- ✔ Слабосвязанные многопроцессорные системы (кластеры) - совокупность компьютеров, объединенных в единую систему быстродействующей сетью и имеющих общую дисковую подсистему;

- ✔ Системы с массовым параллелизмом (MPP) - системы с сотнями и даже тысячами процессоров, имеющие многоуровневую структуру оперативной памяти

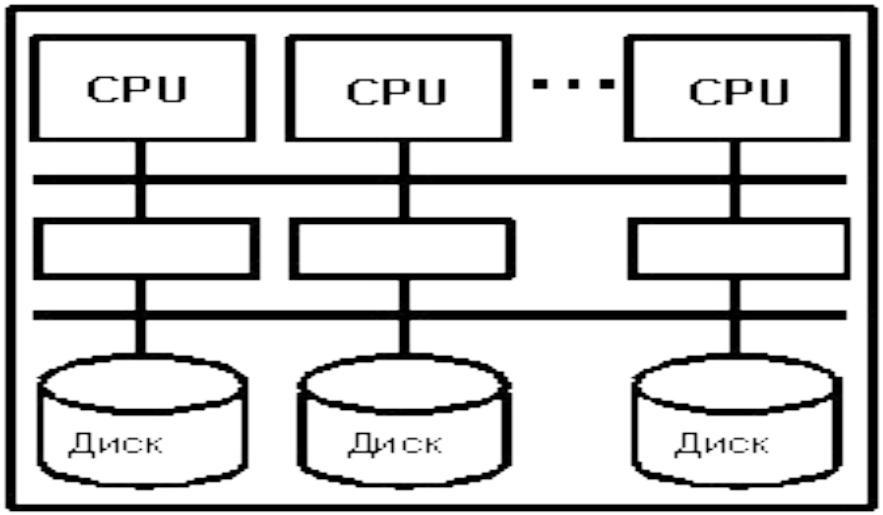
# Параллельные архитектуры серверов баз данных




SMP система




Кластер




MPP система

# Требования, определяющие качества современной СУБД

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

*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 направлена в сторону *объединения задач*: оперативной обработки транзакций (OLTP), поддержки принятия решений (DSS)

*Постоянная доступность данных* реализуется с помощью следующих механизмов:

- ✔ оперативное администрирование;
- ✔ функциональная насыщенность СУБД.

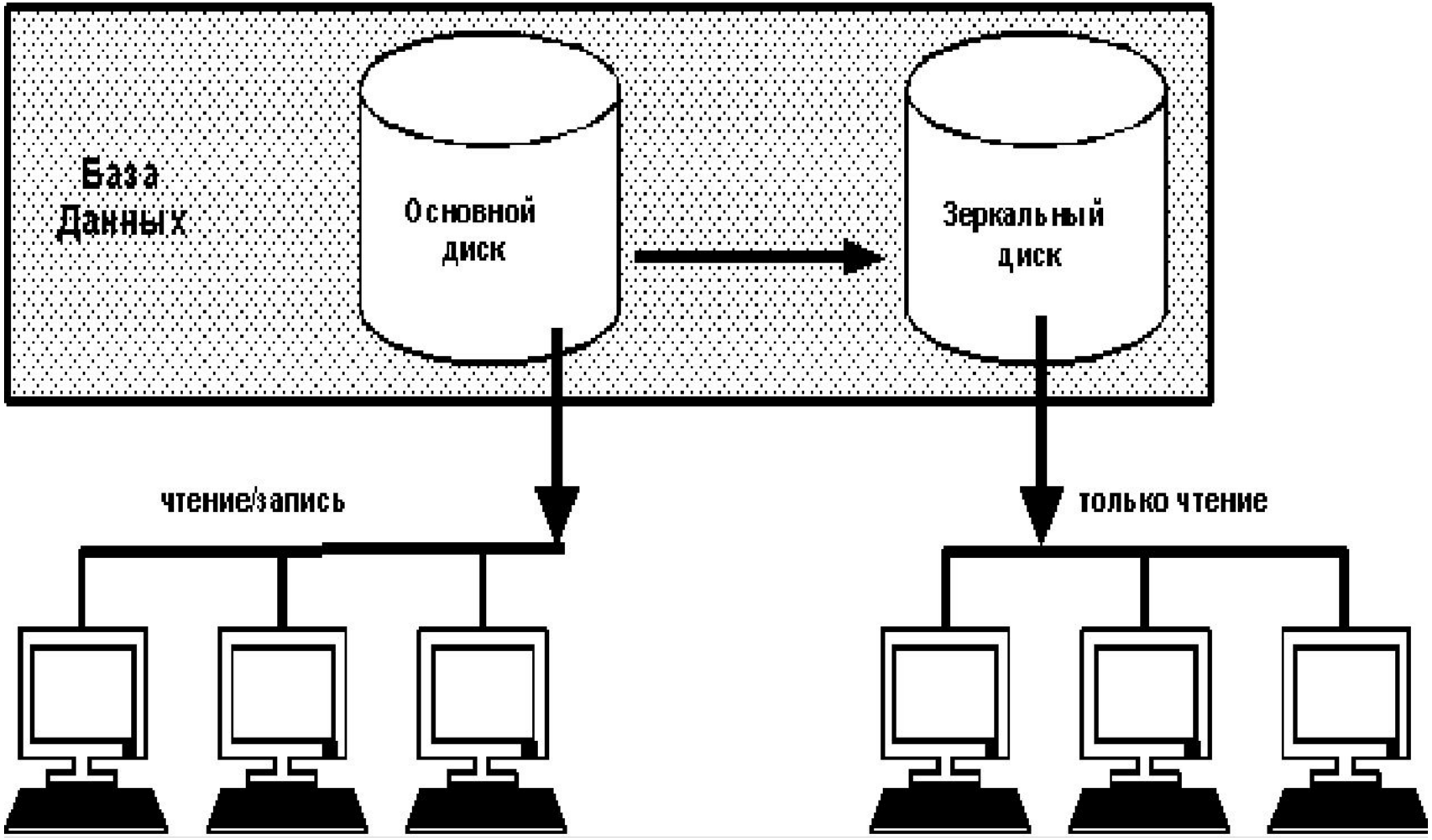
*Утилиты администрирования* призваны поддерживать бесперебойное функционирование СУБД ( сведение к минимуму планируемых или сбойных простоев системы).

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.

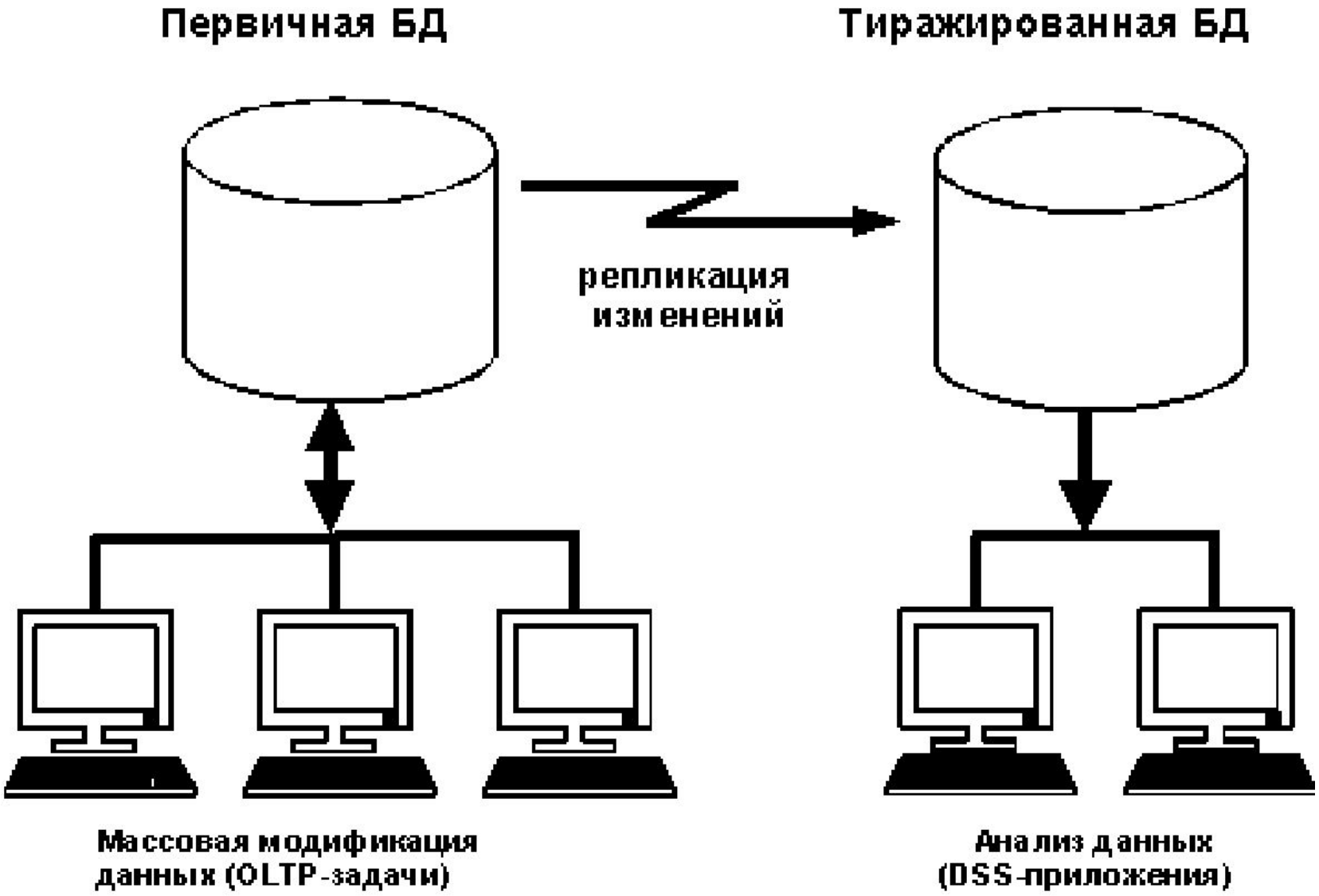
*Управляемая избыточность* данных обычно представлена в двух формах :

- ✔ программное зеркалирование (software mirroring) ;
- ✔ тиражирование (replication) данных.

# Программное зеркалирование (software mirroring)



# Тиражирование (replication) данных



# Средства защиты информации

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 информационной безопасности особенно важны программно-технические меры, т. к. основная угроза компьютерным системам исходит от самих этих систем (сбои оборудования, ошибки программного обеспечения, промахи пользователей и администраторов и т.п.).

# Основные механизмы безопасности

- ✔ Идентификация и аутентификация при помощи паролей;
- ✔ управление доступом (системные привилегии, объектные привилегии);
- ✔ механизм ролей (ROLE);
- ✔ представления (VIEW);
- ✔ триггеры;
- ✔ протоколирование и аудит;
- ✔ криптография;
- ✔ экранирование.

*Аутентификация* - верификация соответствия некоторого субъекта имеющейся априорной информации о нем.

*Авторизация* - предоставление субъекту некоторых прав доступа к информационному объекту.

*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

# Инкапсуляция передаваемой информации в специальных протоколах обмена

***Инфраструктуры с открытыми ключами***. Использование подобных методов 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 этих алгоритмов шифрования заключается в том, что операции шифрования и дешифрования производятся разными ключами (открытым и закрытым соответственно).

Наиболее широко распространены следующие системы : 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.

***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 и др.

# Ограничение информационных потоков

***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.

***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.

При этом методе обращения из глобальной сети в локальную становятся невозможными в принципе.

Также этот метод не дает достаточной защиты против атак на более высоких уровнях - например, на уровне приложения (вирусы, код Java и JavaScript).

***Журнал регистрационной информации*** используется для следующих целей:

- ✔ обнаружения необычных или подозрительных действий пользователей и идентификации лиц, совершающих эти действия;
- ✔ обнаружения попыток несанкционированного доступа;
- ✔ оценки возможных последствий нарушения информационной безопасности;
- ✔ оказания помощи в расследовании случаев нарушения безопасности;
- ✔ организации пассивной защиты от нелегальных действий.

Пользователи, зная, что их действия фиксируются, могут не решиться на незаконные операции.

# 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.

Главная проблема, которую необходимо решать в связи с метками, - это обеспечение их целостности.

# Что будет результатом выполнения следующего кода?

```
set serveroutput on
declare
        a number:=10;
        b number:=0;
begin
        a:=a/b;
exception
        when ZERO_DIVIDE then
        declare
                b number:=1;
        begin
                a:=a/b;
                dbms_output.put_line('Блок выполнен успешно.');
        exception
                when ZERO_DIVIDE then
                dbms_output.put_line('Ошибка: делитель равен нулю.');
        end;
End;
```

***Варианты ответов:***

- ✔ **На экране появится надпись "Блок выполнен успешно"**
- ✔ **На экране появится надпись "Ошибка: делитель равен нулю."**
- ✔ **СУБД выдаст сообщение об ошибке, поскольку в приведенном коде указаны две секции исключений**
- ✔ **На экране ничего не появится, кроме сообщения о том, что процедура успешно выполнена**

# 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.

# Промежуточное программное обеспечение (ППО) 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, 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)* ,
- ✔ 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 в конце концов 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*.

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)

*Технология ODBC* разрабатывалась как общий, независимый от источников данных, способ доступа к данным.

Применение технологии должно было также обеспечить переносимость приложений в среду различных баз данных без потребности переработки самих приложений. В этом смысле технология ODBC уже стала промышленным стандартом, ее поддерживают практически все производители СУБД и средств разработки.

*Основная идея технологии ODBC* :

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

# Этапы процедуры запроса данных через ODBC API

*Этап 1* - *установление соединения*.

Этот этап 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.

*Этап 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.

*Этап 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.

Этот шаг необязателен, если используется связывание столбцов с локальными переменными.

*Этап 4* - *освобождение ресурсов*.

После того, как данные получены, ресурсы освобождаются путем вызова функций освобождения указателей оператора, соединения и среды.

Указатели оператора и соединения могут быть использованы в процессе обработки.

# Oracle8

**Направления развития типов и структур данных:**

- ✔ расширение набора встроенных типов данных;
- ✔ расширение спектра стандартных структур данных;
- ✔ предоставление пользователям возможности определять собственные типы и структуры данных.

**Средства для работы с большими объектами**

Размер большого объекта может достигать 4 ГБ, а сами объекты полноценно участвуют в транзакциях.

Большие объекты представлены как бинарные (BLOB) и символьные (CLOB для текстов типа CHAR и NCLOB — для NCHAR) и могут храниться внешним по отношению к СУБД образом, в файлах операционной системы.

Этот вид хранения обслуживается типом данных BFILE и представлены в столбцах реляционных таблиц LOB-локаторами, содержащими ссылку на реальное место хранения значений.

Операции с большими объектами выполняются средствами пакета DBMS_LOB.

## Идентификаторы реляционных строк

Каждая строка реляционной таблицы в СУБД Oracle8 имеет уникальный идентификатор.

Этот идентификатор имеет тип ROWID.

С каждой реляционной таблицей связан столбец типа ROWID и с именем ROWID, хранящий адреса строк.

Столбец ROWID, доступный только на чтение, может использоваться наравне с другими столбцами таблицы в операторе SELECT и конструкции WHERE.

Для работы с типом ROWID служит пакет DBMS_ROWID.

## Записи

Записи трактуются в языке PL/SQL Oracle8 как совокупность разнотипных (быть может, структурных) компонентов, их можно хранить в столбцах реляционных таблиц, передавать в качестве параметров и т.п.

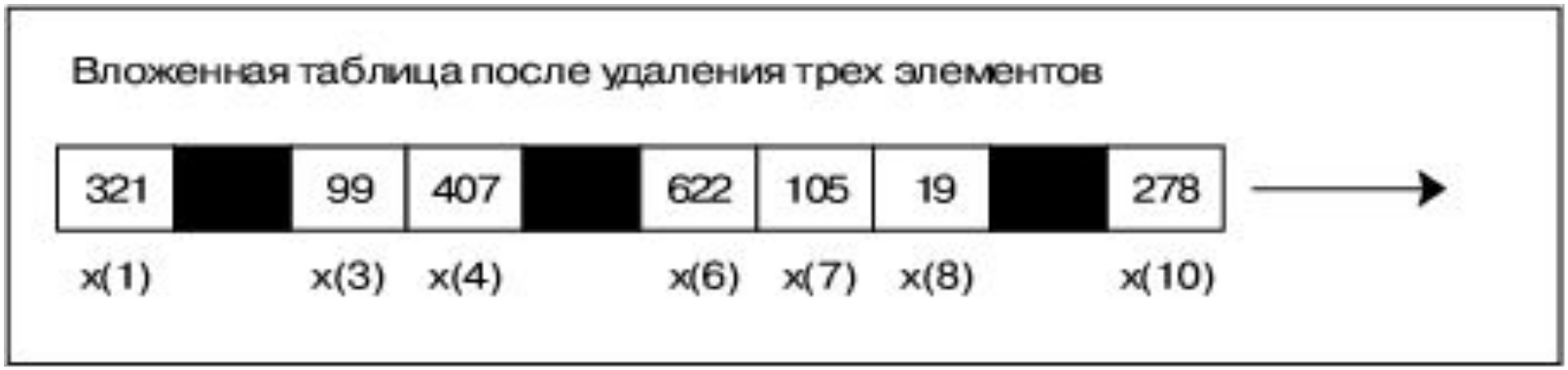
# Коллекции

*Коллекции в Oracle8 представляют собой одномерные массивы с подвижными верхними границами и подразделяются на два вида:*

- ✔ вложенные таблицы (таблицы могут являться атрибутами реляционных таблиц);
- ✔ массивы переменного размера.

Вложенная таблица - обычная реляционная таблица с одним столбцом. Число элементов во вложенной таблице практически не ограничено; существующие элементы могут удаляться, так что таблица не обязана являться непрерывным массивом.

**Вложенная таблица**



# Методы, применимые к коллекциям

- ✔ **EXISTS** (проверяет, существует ли элемент коллекции с заданным номером);
- ✔ **COUNT** (выдает текущее число элементов коллекции);
- ✔ **LIMIT** (выдает NULL для вложенных таблиц и максимальный размер для массивов);
- ✔ **FIRST/LAST** (выдают номер первого/последнего элемента коллекции);
- ✔ **PRIOR/NEXT** (выдают номер предыдущего/следующего элемента коллекции или NULL, если таковой отсутствует);
- ✔ **EXTEND** (добавляет к коллекции заданное число элементов);
- ✔ **TRIM** (удаляет из конца коллекции заданное число элементов);
- ✔ **DELETE** (удаляет заданные элементы вложенной таблицы).

Вызов этих методов оформлен по-объектному принципу: за именем коллекции через точку следует имя метода, затем в скобках — аргументы.

```
TYPE DictItem IS RECORD ( word VARCHAR2 (30),
  traslation VARCHAR2 (200));
TYPE MyDictionary IS TABLE OF DictItem; TYPE
  HisDictionary IS VARRAY (1000)   OF DictItem;
```

```
DECLARE  ...  dict MyDictionary;  ...
BEGIN  ...  IF  dict (i).word = 'Oracle' THEN ...
```

```
FOR  i  IN    dict.FIRST () .. dict.LAST ()
LOOP   IF   dict.EXISTS  (i)  THEN   dict.DELETE (i);
END IF;
END LOOP;
```

# Oracle 9i

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.

Сервер баз данных *Oracle9i предоставляет объектно-реляционную технологию*, которая обеспечивает простые методы разработки, развертывания и управления приложениями, оперирующими со сложными данными.

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).

Можно создать новые типы данных, например,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).

СУБД Oracle9i 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++.

## Объектно-ориентированная разработка приложений

Индустриальные стандарты для разработки объектно-ориентированных приложений:

- ✔ **UML (Unified Modeling Language)** – унифицированный язык моделирования для объектно-ориентированного анализа и проектирования; (новая версия UML 2.0 в 2003 г.)
- ✔ **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.

# Объектные типы

Корпорация Oracle расширила SQL, который позволяет 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 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.*

## Синонимы типов

Точно так, как можно создавать синонимы таблиц, представлений и других различных объектов схемы, можно создавать синонимы определяемых пользователями типов.

Синонимы этих типов предоставляют независимый от местонахождения способ ссылки на объекты схемы. Приложение, использующее синонимы общедоступных типов, можно развернуть без какого-либо изменения любой схемы базы данных. При этом не нужно квалифицировать имя типа именем схемы, в которой этот тип был определен.

## Эволюция типов

Использование типа позволяет пользователю развивать бизнес-логику, зафиксированную в поведении этого типа. Эволюция типов– это механизм, который позволяет пользователю изменять тип и распространять эти изменения на другие объекты схемы, которые ссылаются на модифицированный тип. К числу объектов схемы, которые могут ссылаться на тип, относятся другие типы, подтипы, объекты-строки, объекты-столбцы (column objects), программные блоки (пакеты, функции, процедуры), представления, функциональные индексы и триггеры.

## Поддерживаемые операции эволюции типов

*Операции над атрибутами типов*:

- ✔ добавление атрибута для типа;
- ✔ удаление атрибута типа;
- ✔ модификация типа атрибута (увеличение его длины, точности или масштаба).

*Операции над методами типов*:

- ✔ добавление метода для типа;
- ✔ удаление метода типа.

Изменение свойств INSTANTIABLE (абстрактный/не абстрактный тип – не допускает или допускает непосредственное порождение экземпляров объектов) и FINAL (терминальный/нетерминальный тип – нельзя создавать подтипы или можно) объектного типа SQL.

Поддержка явного распространения изменений типа на зависимые от него типы и таблицы.

Эти изменения при эволюции типов являются либо *структурными* (structural), либо *неструктурными (*non-structural).

# 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 включена поддержка XML. В сервер Oracle интегрированы средства поддержки OLAP и добычи данных.

# Надежность и масштабируемость

## *Real Application Cluster (RAC)*

Достижением Oracle 9i в области обеспечения высокой надежности и масштабируемости стали 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.

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.

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 *10g предлагает 10 важнейших усовершенствований*:

- ✔ рекордное повышение производительности;
- ✔ Самоуправляемость;
- ✔ автоматическое управление хранением и доступом к данным (ASM);
- ✔ обновление программного обеспечения 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 10g

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 10g 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 клиентам и партнерам компании снизить затраты и сложности, сопряженные с администрированием бизнес-приложений, благодаря автоматизированному 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.

# Oracle 11g

**Развитие СУБД Oracle как платформы для GRID вычислений**

Начиная с версии 10g компания Oracle позиционирует свою СУБД как платформу для GRID вычислений.

Концепция GRID вычислений достаточно проста, понятна, гибка и позволяет экономить средства предприятия. Поэтому в последнее время наблюдается постепенное внедрение этой архитектуры в IT инфраструктуру.

Вскоре ожидается появление первых GRID с более чем тысячью процессоров.

Сегодня многие крупные информационные системы используют от 100 до 300 процессоров.

Реализуются как малые кластеры, состоящие из нескольких больших SMP машин (например, 2 узла по 64 процессора или 4 узла по 32 процессора), так и большие кластеры, состоящие из множества мелких элементов (например, 32 узла по 4 процессора).

Для российских заказчиков тестируются конфигурации с 1,5 сотней процессоров.

Все большую популярность приобретают многоядерные процессоры. Сейчас большинство новых серверов имеет двух – четырех ядерные процессоры и это не предел.

Нужна платформа, позволяющая эффективно реализовывать приложения на этой инфраструктуре. Oracle предлагает в качестве такой платформы GRID на основе СУБД Oracle 11g.

Уже сегодня *Oracle 10g позволяет объединить в кластер до 64 узлов*.

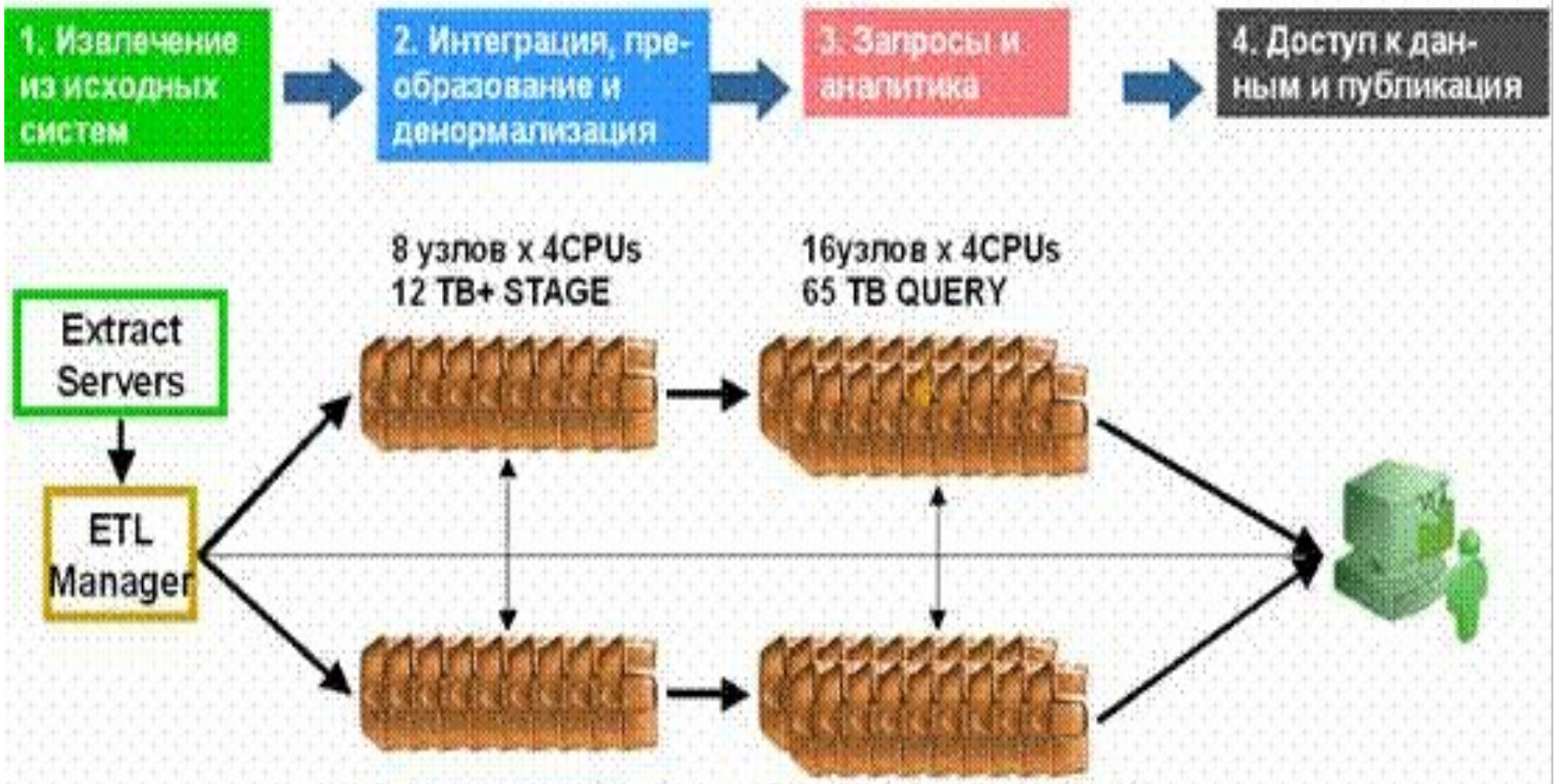
*В Oracle 11g эта цифра удваивается*. И каждый узел может иметь множество процессоров и ядер.

Таким образом, суммарная вычислительная мощность такой GRID может превысить вычислительную мощность серьезных mainframe машин. Кстати, уже сегодня Oracle использует для TPC тестов двухтеррабайтный буферный кэш, так что растет не только процессорная мощность GRID, но и ее суммарная память.

Одним из примеров удачного внедрения GRID технологии является хорошо известный многим интернет-магазин Amazon.com. Изначально хранилище данных Amazon.com было реализовано на основе нескольких SMP машин, но затем, с целью повышения мощности и снижения стоимости системы, ее решили перевести на платформу GRID. В качестве элементов GRID использовались четырехпроцессорные компьютеры с ОС Linux, на которых был установлен Oracle 10g RAC и Oracle ASM. Архитектура системы  на рисунке .

# Amazon DW архитектура на основе Oracle 10g RAC и ASM на Linux

Кластер Amazon настолько эффективен и дешев, что они реализовали 2 одинаковых линейки и еще сэкономили деньги.




2 линейки одинаковых RAC кластеров позволяют отказаться от backup активных оперативных данных

***Система состоит из нескольких блоков*:**

- ✔ извлечение данных из исходных систем
- ✔ интеграция, преобразование и денормализация данных
- ✔ блок обработки запросов и анализа данных
- ✔ блок доступа к данным и публикации.

Извлечением данных занимаются так называемые *extract серверы*.

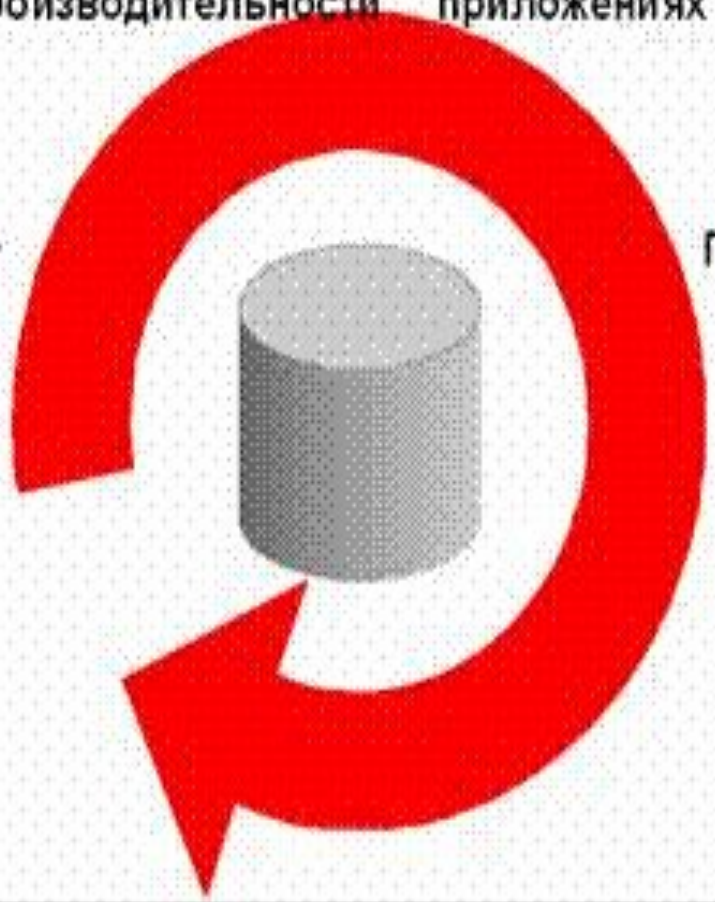
Далее они передают данные в блок интеграции и преобразований. SMP машины блока интеграции и преобразований были заменены GRID из 8 узлов. Объем данных, хранимых на этом этапе – 12 терабайт. SMP машины блока обработки запросов и анализа были заменены на GRID из 16 узлов. Объем данных хранилища – 66 терабайт.

Данные extract серверов поступают в первый GRID (это Stage область), после чего загружаются в хранилище на второй GRID.

После реализации такой линейки из 8+16=24 узлов выяснилось, что стоимость такой инфраструктуры, благодаря использованию дешевых элементов, более чем в 2 раза ниже стоимости предыдущей системы. Поэтому было принято решение реализовать вторую такую же линейку из 8+16 узлов, которая будет дублировать работу первой линейки. Теперь данные extract серверов поступают как на первую, так и на вторую линейку серверов и в компании всегда существует 2 одинаковые версии хранилища. Одна из них является основной, а на вторую можно переключиться в случае сбоя. Такая архитектура позволила отказаться от частого копирования активных оперативных данных. Причем суммарная стоимость такой продублированной архитектуры оказалась ниже стоимости старой системы.

Особое внимание было *обращено на снижение времени простоя элементов GRID*. Если раньше большинство производителей СУБД прилагало усилия к снижению времени простоя систем, возникающего из-за внезапных, незапланированных причин (поломка компьютера, сбой операционной системы или приложения, человеческие ошибки, катастрофы, потери файлов и т д.), то теперь Oracle сосредоточился на снижении времени плановых простоев.

# Новые возможности Oracle Database 11g для сохранения работоспособности при внесении изменений



# Хранилища данных (Data Warehouse) и оперативный анализ данных (On-Line Analytical Processing, OLAP)

- ✔ новые информационные технологии, которые обеспечивают бизнес-аналитикам и управленцам возможность изучать *большие объемы данных* при помощи *быстрого интерактивного отображения* информации на *разных уровнях детализации*.

- ✔ одно из наиболее динамично развивающихся направлений индустрии создания программного обеспечения.

- ✔ Большинство ведущих производителей программ, включая *Arbor Soft-ware*, *Cognos*, *IBM*, *Informix*, *Microsoft*, *Oracle*, *SAS Institute*, *Sybase*, ведут конкурентную борьбу в данном секторе рынка.

- ✔ Однако ни одна из фирм-разработчиков программного обес печения OLAP не поставляет законченного решения для корпоративной аналитической системы, их продукты являются лишь набором инструментальных средств, на базе которых строятся системы поддержки принятия решений.

# Хранилище данных (Data Warehouse)

Создание хранилищ данных - процесс сбора, отсеивания и предварительной обработки данных с целью предоставления результирующей информации пользователям для статистического анализа (создания аналитических отчетов).

Областями приложения рассмотренных программных продуктов могут быть и физика, и химия, и биология, и медицина, и экология, и сложные технические объекты, и любая другая область исследовательской деятельности, которая связана с *анализом больших систем.*

*Хранилище данных – место, где люди могут получить доступ к своим данным* (*Ральф Кимбалл*)

Большинство имеющихся систем ХД созданы для банков и кредитных учреждений.

# Хранилище Данных

- ✔ предметно-ориентированный,
- ✔ интегрированный,
- ✔ зависимый от времени набор данных,
- ✔ предназначенный для поддержки принятия решений различными группами пользователей.

Хранилище носит *предметно-ориентированный характер*, его организация нацелена на *содержательный анализ* информации, а не на автоматизацию бизнес-процессов.

Это свойство определяет архитектуру построения хранилища и принципы проектирования модели данных, отличные от тех, что применяются в оперативных системах.

# Хранилище Данных

***Интегрированность*** означает, что, например, данные о клиентах, подразделениях и банковских продуктах, полученные из различных источников, хранятся *согласованно и централизованно*.

При этом полная информация о клиенте может включать данные, поступившие как из основной автоматизированной банковской системы (АБС), так и из фронтофисного или иного приложения.

Хранилище содержит ***исторические данные***, или зависимый от времени набор данных.

Если в оперативных источниках представлены самые последние значения (текущее наименование клиента или его физический адрес), то ХД будет содержать в себе всю их предысторию с указанием периода, когда те или иные данные были актуальны.

*ХД предназначено для поддержки принятия решений*, и его пользователи — это высший и средний менеджмент предприятия, аналитики, представители подразделений финансового анализа и маркетинга.

# Предпосылки создания ХД

**Ужесточение конкуренции**

отсутствием высокодоходных финансовых инструментов и невысокой средней нормой прибыли

**Развитие систем управления взаимоотношениями с клиентами (CRM)**

Для построения эффективной стратегии таких взаимоотношений необходимо хранилище данных, с помощью которого легко определить, какой клиент является наиболее прибыльным и выгодным для предприятия. Это даст любому кредитному учреждению возможность выработать единую и эффективную политику по отношению к каждому клиенту.

**Разрозненность данных**

неизбежно присутствуют разнородные источники информации.

Отдельные системы автоматизации позволяют получить отчет по определенной группе смежных банковских продуктов (чаще всего они отражают бухгалтерскую прибыль), этих данных недостаточно для управления бизнесом.

# Требования к хранилищам данных

- ✔ Нацеленность на ключевые понятия, а не процессы;
- ✔ поддержка высокой скорости получения данных из хранилища;
- ✔ поддержка внутренней непротиворечивости данных;
- ✔ возможность получения и сравнения так называемых срезов данных (slice and dice);
- ✔ наличие удобных утилит просмотра данных в хранилище;
- ✔ полнота и достоверность хранимых данных;
- ✔ поддержка качественного процесса пополнения данных;
- ✔ простое и удобное взаимодействие пользователей с системой;
- ✔ для представления данных в удобном для анализа виде применяются OLAP-технологии.

OLAP является аналитическим инструментом и одним, но далеко не единственным средством анализа данных в хранилище. Важно отметить, что средства OLAP могут быть использованы и вне хранилища.

OLAP-анализ данных, находящихся в своих источниках, может быть произведен без их извлечения и загрузки в хранилище. Однако эффективность многомерного анализа при наличии хранилища данных резко возрастает.

# Структура СППР с физических ХД


*При загрузке в ХД данные фильтруются.*

Во время загрузки данные очищаются и приводятся к единому формату.

*В ХД хранится обобщенная информация, которая отсутствует в ОИД.*

# Отличия хранилищ данных от баз данных

- ✔ обычные базы данных предназначены для того, чтобы помочь пользователям выполнять повседневную работу, тогда как хранилища данных предназначены для принятия решений;

- ✔ обычные базы данных подвержены постоянным изменениям в процессе работы пользователей, а хранилище данных относительно стабильно: данные в нем обычно обновляются согласно расписанию;

- ✔ обычные базы данных чаще всего являются источником данных, попадающих в хранилище. Кроме того, хранилище может пополняться за счет внешних источников, например, статистических отчетов.

# OLAP
# (On-Line Analytical Processing)

- ✔ Концепция OLAP была описана в 1993 году Эдгаром Коддом.
- ✔ OLAP - технология комплексного многомерного анализа данных.
- ✔ Основана на построении многомерных наборов данных – OLAP-кубов, оси которого содержат параметры, а ячейки – зависящие от них агрегатные данные.
- ✔ OLAP — это ключевой компонент организации хранилищ данных.

# Приложения OLAP

Приложения с OLAP-функциональностью должны:

- ✔ предоставлять пользователю результаты анализа за приемлемое время,
- ✔ осуществлять логический и статистический анализ,
- ✔ поддерживать многопользовательский доступ к данным,
- ✔ осуществлять многомерное концептуальное представление данных
- ✔ иметь возможность обращаться к любой нужной информации.

# Приложения OLAP

Многомерность в OLAP-приложениях может быть разделена на три уровня:

- ✔ *Многомерное представление данных* - средства конечного пользователя, обеспечивающие многомерную визуализацию и манипулирование данными; слой многомерного представления абстрагирован от физической структуры данных и воспринимает данные как многомерные.
- ✔ *Многомерная обработка* - средство (язык) формулирования многомерных запросов (традиционный реляционный язык SQL здесь оказывается непригодным) и процессор, умеющий обработать и выполнить такой запрос.
- ✔ *Многомерное хранение* - средства физической организации данных, обеспечивающие эффективное выполнение многомерных запросов.

# Основные элементы хранилищ данных

***Оперативные источники данных***

разнородные источники данных, слабоструктурировнные

***Процедура загрузки данных***

экспоненциально зависит от количества источников

(включает в себя очистку, согласование и контроль качества)

***Отраслевая модель данных***

Хранилище данных может быть реализовано как на реляционной, так и на многомерной СУБД. Но, как показывает практика, хранилища серьезного объема реализованы в основном на реляционных СУБД. Центральным компонентом хранилища является отраслевая модель данных, и ее тщательная проработка во многом определяет успешность проекта в целом.

***Витрины данных***

Витрины, построенные на основе ХД или на базе первичных источников, проектируются для удовлетворения потребностей определенной группы пользователей, ориентированных на решение конкретных аналитических задач.

***Представление данных и способы их анализа***

*интерактивный анализ данных (Online Analytical Processing, OLAP)* — компьютерное приложение, поддерживающее многомерное представление и визуализацию данных с целью их анализа и подготовки отчетов;

*периодически выпускаемая отчетность (Reporting)* — отчеты в стандартных формах;

*интеллектуальный анализ данных (Data Mining)* — процесс анализа больших наборов данных, применяемый для обнаружения связей между различными их элементами и поиска скрытых закономерностей.

# Структура хранилищ данных

типичная структура хранилища данных существенно отличается от структуры обычной реляционной СУБД – как правило, *она денормализована* (это позволяет повысить скорость выполнения запросов) ;

основными составляющими структуры хранилищ данных являются *таблица фактов* (fact table) и *таблицы измерений* (dimension tables).

***В основе концепции ХД лежит идея разделения данных, используемых для оперативной обработки и задач анализа***.

Такое разделение позволяет оптимизировать как *структуру данных* оперативного хранения (оперативные БД, файлы, элек.таблицы)для выполнения операция ввода, модификации и удаления и поиска, так и струткуру данных, используемых для анализа (для выполнения аналитических запросов).

# Организация Хранилища Данных

Данные в ХД делятся *на три основных категории:*

- *Детальные данные* (переносимые из источников),
- *Агрегированные данные* (обобщение путем суммирования фактических данных по определен. измерениям),
- *Метаданные* (информация о содержащихся в Хранилище данных).
  - Описание объектов.
  - Описание пользователей.
  - Описание места хранения.
  - Описание действий над данными.
  - Причины, повлекшие выполнения над данными тех или иных операций.

# Обобщенная архитектура системы поддержки принятия решений

# Исследования в области баз данных

Сообщество разработчиков баз данных предлагает расширить спектр исследований, приступив к решению вопросов получения, хранения, анализа и представления огромных объемов оперативной информации.

Три основных фактора определят *направленность исследований*:

- ✔ Web существенно облегчает размещение информации в киберпространстве и обеспечивают ее доступность почти для каждого.
- ✔ Постоянное усложнение прикладных сред увеличивает потребность в интеграции программ и данных.
- ✔ Достижения в области компьютерных архитектур делают недействительными предположения и проектные решения, положенные в основу современной технологии СУБД.

## *Программа исследований*

- ✔ СУБД в стиле «plug and play».
- ✔ Объединение миллионов баз данных.
- ✔ Переосмысление традиционной архитектуры систем баз данных.
- ✔ Унификация процессов и данных.
- ✔ Интеграция структурированных и полуструктурированных данных.

*Основная цель для исследователей баз данных*: облегчить для каждого хранение, организацию, доступ и анализ максимума накопленной информации в оперативном режиме.

Большая часть накопленной человечеством информации будет располагаться в WWW. Во многих форматах по всей планете будут рассредоточены экзабайты данных.

Без новых инструментов находить и понимать ответы на наши вопросы будет еще труднее, чем сегодня, поэтому грандиозной целью является превращение Web в следующем десятилетии в более полезную информационную службу.