Объектно-реляционные базы данных

История вопроса

Начало пути:

- ✔ Декабрь 1996 года – компания Informix выпустила объектно-реляционную систему управления базами данных (ОРСУБД) Informix Universal Server.
- ✔ 1997 г. на рынке появились ОРСУБД компаний Oracle (Oracle8) и IBM (DB2 Universal Database).

Постулат того времени: новая парадигма в корне изменит способы проектирования и разработки приложений баз данных!

До конца 1990-х гг. Informix, Oracle и IBM совершенствовали свои ОРСУБД.

1999 г. – появился стандарт SQL:1999, в котором были зафиксированы объектные расширения языка SQL.
2003 г. – выход стандарта SQL:2003, уточнившего и дополнившего SQL:1999.

И тишина...

К разговору об определениях

- Что такое ОРСУБД?
- Что такое объектно-реляционная база данных?

Банальное толкование:

ОРСУБД – традиционная реляционная СУБД, расширенная основными объектными возможностями.

Возражения:

1) Имеющиеся сегодня на рынке ОРСУБД не являются «традиционными реляционными», поскольку в них не поддерживается реляционная модель данных. Они основаны на другой модели данных, представленной в стандарте языка SQL.
2) Объектный мир определен в целом настолько расплывчато и нечетко, что невозможно однозначно говорить об основных объектных возможностях.

Сегодня под ОРСУБД следует понимать системы, которые следуют духу *Манифеста систем баз данных третьего поколения* и букве стандартов SQL:1999 и SQL:2003.

В качестве примеров реализаций ОРСУБД можно использовать Oracle (начиная с Oracle 9i) и IBM DB2 (начиная с версии 7).

Первые реализации: Informix Universal Server (IUS)

- **Интеллектуальные большие объекты.**
  Отличие от BLOB и CLOB реляционных систем. Журнализация и откат изменений.
- **Определение новых базовых типов данных.**
  - **Индивидуальные типы**
    Индивидуальный тип данных (distinct type) основан на существующем типе, но обеспечивает автоматическое преобразование к нужному значению (за счет явного определения соответствующих функций).
  - **Типы со скрытой структурой**
    Типы со скрытой структурой (*opague type*) – абстрактные в строгом смысле этого слова. СУБД IUS могла манипулировать соответствующими значениями только посредством предоставленных разработчиком функций.
  - **Специальные методы хранения, поиска и индексации**
    Определение новых базовых типов данных одновременно с введением специальных алгоритмов хранения, доступа и индексирования, которые отличались от стандартных алгоритмов, реализованных в сервере: определить набор серверных функций, реализующий для нового типа алгоритмы доступа, просмотра, выделения памяти и т. д. (определить на языке С и скомпилировать в объектный формат); описать новый базовый тип данных и указать функции, реализующие для этого типа алгоритмы извлечения и записи на диск значений данного типа.

Первые реализации: Informix Universal Server (IUS)

- **Составные типы данных**

Составные типы могли иметь структуру записи, множества, мультимножества или списка. Структуры данных могли быть вложенными, то есть значениями столбцов таблиц могут быть записи, множества или списки, состоящие из атомарных или составных значений.

Тип данных со структурой записи

CREATE ROW TYPE <имя типа> (<имя поля> <тип поля>, …)

Составной тип можно было использовать для определения столбцов таблицы. Для доступа к отдельным полям значений типа записи использовалась традиционная точечная нотация. Кроме того, типы со структурой записи можно было использовать для определения *типизированных таблиц* с помощью оператора

CREATE TABLE <имя таблицы> OF TYPE <имя типа>;

У типизированной таблицы в IUS имелся один столбец соответствующего составного типа.
К типам записей и таблицам в IUS мог применяться механизм наследования.
Поддерживалось только одиночное наследование.
Все функции, определенные для супертипа, автоматически распространялись на все его подтипы. Для подтипов могли определяться свои функции, как новые, дополняющие функциональность, так и модифицированные, изменяющие функциональность, унаследованную от предка. В последнем случае имело место перекрытие функций и требовалось отложенное связывание.

Первые реализации: Informix Universal Server (IUS)

Наследование таблиц являлось развитием концепции наследования типов. В иерархии наследования могли участвовать только типизированные таблицы, типы которых (типы записи) образуют параллельную иерархию.

Кроме столбцов супертаблиц, подтаблицы наследовали ограничения целостности (первичные ключи, уникальность, ссылочные ограничения), опции хранения, триггеры, индексы и методы доступа.

При определении подтаблиц некоторые свойства супертаблиц можно было переопределять.

Отношение наследования между таблицами являлось динамическим в том смысле, что, если изменялись наследуемые характеристики супертаблиц, то это немедленно отражалось и на подтаблицах (как на прямых наследниках, так и отделенных несколькими уровнями иерархии).

В IUS операции SELECT, UPDATE и DELETE, примененные к супертаблице, распространяются также и на все ее подтаблицы.

Если требовалось ограничить запрос ровно одной супертаблицей, следовало употребить в запросе конструкцию ONLY.

Первые реализации: Informix Universal Server (IUS)

- **Типы коллекций**

Три вида типов коллекций: множеств, мультимножеств и списков.

Тип множества определялся с помощью конструкции

CREATE TYPE <имя типа> SET <тип элементов> NOT NULL

Тип элементов множества мог задаваться именем типа для предопределенных или ранее определенных типов данных или определяться «по месту» для множеств, состоящих из элементов типа записи.

Тип мультимножества определялся с помощью конструкции

CREATE TYPE <имя типа> MULTISET <тип элементов> NOT NULL;

Использование типа множества и мультимножества – после предиката IN в разделе WHERE оператора SELECT.

К этим значениям была применима функция CARDINALITY, возвращающая число элементов. Средствами прямого SQL можно было осуществлять доступ к значениям-множествам (и мультимножествам) только как к единому целому. Доступ к элементам множества был возможен только при использовании встраиваемого SQL IUS или языка определения хранимых процедур (SPL). Основная идея: множество представлялось в виде производной таблицы.

Первые реализации: Informix Universal Server (IUS)

Список в IUS являлся упорядоченным набором элементов, в котором допускалось наличие дубликатов. Список отличается от мультимножества тем, что для каждого элемента списка имеется порядковый номер (нумерация начинается с единицы).
Определение типа списка выглядит следующим образом:

CREATE TYPE <имя типа> LIST <тип элементов> NOT NULL

Модуль DataBlade – это функционально законченное расширение сервера IUS, рассчитанное на определенное приложение или класс приложений.
С модулями DataBlade не были связаны какие-то особые возможности сервера или языковые конструкции. Такие модули строились из компонентов, описанных выше, то есть из определяемых разработчиком подпрограмм, типов и структур данных, методов доступа и других объектов, хранящихся в базах данных. Создание и сопровождение модулей DataBlade в IUS поддерживалось путем предоставления соответствующего инструментария – среды разработки, «упаковщика» модулей, утилиты регистрации, прикладного программного интерфейса.

Первые реализации: Oracle 8

- **Объектные типы и объектные таблицы**

Объектные типы в Oracle8 являлись аналогом типа записи в IUS. Объектный тип данных определялся в следующем синтаксисе:

CREATE TYPE <имя типа> AS OBJECT (<имя поля> <тип поля>, …);

Как и в IUS, для доступа к отдельным полям значений объектных типов использовалась точечная нотация.

- **Методы объектных типов**

Методы представляли собой функции, написанные на PL/SQL, Java, C или другом языке и сохраненные в базе данных или вне ее (при наличии регистрации в базе данных).

Методы объектного типа разбивались на три категории: методы-члены, статические методы и методы сравнения.

Методы-члены вызывались в нотации

<имя_объекта>.<имя_метода> (список_параметров).

У них имелся неявный параметр SELF, и они могли обращаться к значениям атрибутов объекта. Под термином *объект* здесь понимается строка объектной таблицы. Как правило, *именем объекта* являлось имя объектной таблицы или ее псевдонима, используемые в операторе выборки SQL.

Первые реализации: Oracle 8

Статические методы вызывались в нотации

<имя_объектного_типа>.<имя_метода> (список_параметров)

и не могли обращаться к значениям атрибутов конкретных объектов.

Методы сравнения служили для сравнения экземпляров объектов:

- методы вида MAP – принимает в качетсве параметра объект некоторого объектного типа, а возвращает значение одного из встроенных типов, которое может использоваться в операциях сравнения и сортировки;
- методы вида ORDER – сравнивает два объекта и возвращает *–1*, если первый объект меньше второго, *0*, если объекты равны, и *1*, если первый объект больше второго.

Метод-конструктор – это функция, которая возвращает объект данного типа. Имя метода-конструктора совпадает с именем объектного типа, имена и типы параметров конструктора – с именами и типами атрибутов объектного типа.

Первые реализации: Oracle 8

Объектные таблицы, ссылочные типы

Объектной таблицей в Oracle8 называлась таблица, строки которой имеют объектный тип. Синтаксис определения близок к синтаксису в IUS.

В Oracle8 не поддерживалось наследование (ни типов, ни таблиц), однако уже в Oracle8i (начало 1998 г.) появилась возможность наследования таблиц.

В Oracle8 строки объектных таблиц назывались *строчными объектами* (*row objects*). Для всех строчных объектов обеспечивалась возможность уникальной идентификации. Использовалось понятие *объектного идентификатора*, и столбец любой таблицы можно было определить на встроенном типе REF. Значения этого столбца являлись своего рода указателями на строчные объекты той же самой или другой объектной таблицы.

В качестве значений типа REF использовались уникальные идентификаторы строк таблиц RowID, которые в Oracle почти напрямую адресуют строки во внешней памяти.

В SQL Oracle8 поддерживалась встроенная функция REF, позволяющая получить значение объектного идентификатора указанной строки.

Первые реализации: Oracle 8

Типы коллекций в Oracle8

- *табличные типы (table types);*
- *типы массивов (array types).*

Значениями каждого типа коллекции являлись коллекции (таблицы или массивы) элементов одного и того же типа (типа элемента). Тип элемента мог быть встроенным или объектным типом, но не типом коллекции.

Табличный тип

Создавался конструкцией следующего вида:

CREATE TYPE <имя типа> AS TABLE OF <тип элемента>;

После этого можно было определить таблицу, типом одного из столбцов которого являлся данный табличный тип. В действительности для хранения табличных значений этого столбца создавалась одна дополнительная «вложенная» таблицы, строки которой связывались с соответствующими строками внешней таблицы с помощью объектных идентификаторов.

Такая организация позволяла выбирать связанные данные из внешней и вложенной таблиц без использования явной операции соединения и достаточно эффективно за счет поддержки явных связей между строками.

Первые реализации: Oracle 8

Типы коллекций в Oracle8

Тип массива *- VARRAY*

Это массив переменного размера (varying-length array).

Для определения типа массива использовалась следующая синтаксическая конструкция:

CREATE TYPE <имя типа> AS VARRAY (<натуральное число>)
OF <тип элемента>;

В отличие от значений табличного типа, для элементов значений типа массива поддерживается порядок.

Как и в случае с множествами (и мультимножествами) в IUS, на уровне прямого SQL Oracle8 можно было работать только с массивами целиком. Однако существовала возможность привести в запросе (или другом операторе SQL) тип массива к табличному типу.

Манифест систем баз данных 3-го поколения

I поколение: иерархические и сетевые БД (70-е гг. XX века).
II поколение: реляционные БД (80-е гг. XX века)
III поколение: постреляционные БД (с середины 90-х годов XX века по наст. время).

Примеры сложных предметных областей:
системы автоматизации проектирования (САПР), системы CASE и гипертекстовые приложения и т.п.

Принципы СУБД третьего поколения:

1. Помимо традиционных услуг по управлению данными, СУБД третьего поколения обеспечат поддержку более богатых структур объектов и правил.
2. СУБД третьего поколения должны включить в себя СУБД второго поколения.
3. СУБД третьего поколения должны быть открыты для других подсистем.

Предложения манифеста

1. **Предложения, касающиеся управления объектами и правилами**

Предложение 1.1: Система типов СУБД третьего поколения должна быть богатой и разнообразной.

Все перечисленное является желательным:

1) система абстрактных типов данных для создания новых базовых типов

2) конструктор типа массив

3) конструктор типа последовательность

4) конструктор типа запись

5) конструктор типа множество

6) функции как тип

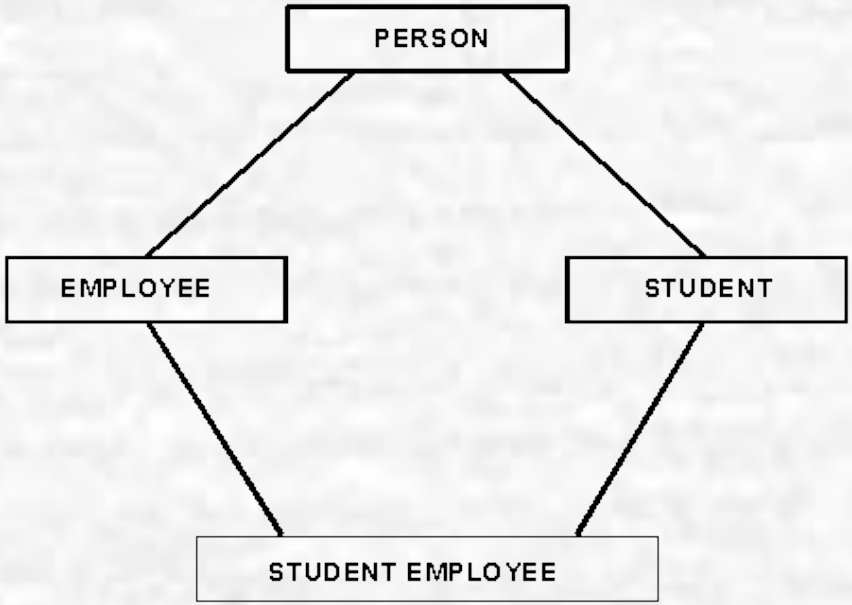
7) конструктор типа объединение

8) рекурсивная композиция всех перечисленных выше конструкторов

Предложения манифеста

1. **Предложения, касающиеся управления объектами и правилами**

Предложение 1.2: Наследование - хорошая идея.



Предложение 1.3: Функции, в том числе процедуры и методы баз данных, и инкапсуляция - хорошие идеи.

Предложения манифеста

1. **Предложения, касающиеся управления объектами и правилами**

Предложение 1.4: Уникальные идентификаторы (UID) записей должны задаваться СУБД только в том случае, когда недоступен определенный пользователем первичный ключ.

Предложение 1.5: Правила (триггеры, ограничения) станут одной из ключевых характеристик будущих систем. Их не следует ассоциировать с определенными функциями или наборами.

2. Предложения, касающиеся увеличения функциональных возможностей СУБД

Предложение 2.1: Все программируемые доступы к базам данных должны осуществляться через непроцедурный язык доступа высокого уровня.

Предложение 2.2: Должно быть по крайней мере два способа спецификации наборов: посредством перечисления членов и путем использования языка запросов для задания членов.

Предложения манифеста

2. Предложения, касающиеся увеличения функциональных возможностей СУБД

Предложение 2.3: Существенно наличие обновляемых представлений.

Предложение 2.4: Показатели производительности не имеют почти ничего общего с моделями данных и не должны в них проявляться.

Основными параметрами, по которым оценивается производительность работы с использованием как SQL, так и спецификаций более низкого уровня, являются:

- объем работ по оптимизации настройки СУБД с целью повышения ее эффективности
- использование в СУБД методов компиляции
- местонахождение буферного пула (в адресном пространстве клиента или СУБД)
- доступные типы индексирования
- производительность интерфейса клиент СУБД
- поддерживаемая кластеризация

Предложения манифеста

3. Предложения, следующие из необходимости открытости системы

Предложение 3.1: СУБД третьего поколения должны быть доступны из различных ЯВУ.

Предложение 3.2: Язык "X с поддержкой стабильных данных" (для различных X) - хорошая идея. Языки будут поддерживаться над единой СУБД благодаря расширениям компилятора и (более или менее) сложной системе времени выполнения.

Предложение 3.3: Хорошо это или плохо, но SQL становится интергалактическим языком данных.

Предложение 3.4: Запросы и ответы на них должны образовывать нижний уровень коммуникаций между клиентом и сервером.

Oracle: пример использования вложенных таблиц

Определим объект типа *Course*:

```
SQL> CREATE TYPE Course AS OBJECT (
    course_no NUMBER(4),
    title     VARCHAR2(35),
    credits    NUMBER(1));
```

Определим TABLE type *CourseList*, который содержит объекты *Course*:

```
SQL> CREATE TYPE CourseList AS TABLE OF Course;
```

Создадим таблицу *department*, которая содержит столбец типа *CourseList*:

```
SQL> CREATE TABLE department (
    name     VARCHAR2(30),
    director VARCHAR2(40),
    office   VARCHAR2(40),
    courses  CourseList)
    NESTED TABLE courses STORE AS courses_tab;
```

Oracle: пример использования вложенных таблиц

Пример заполнения таблицы:

```
INSERT INTO department
VALUES('Psychology', 'Irene Friedman', 'Fulton Hall 133',
    CourseList(Course(1000, 'General Psychology', 5),
    Course(2100, 'Experimental Psychology', 4),
    Course(4320, 'Cognitive Processes', 4),
    Course(4410, 'Abnormal Psychology', 4)));
```

С этой таблицей можно работать программно:

```
DECLARE
      new_courses CourseList := CourseList(Course(1002, 'Expository Writing', 3),
            Course(2020, 'Film and Literature', 4),
            Course(2810, 'Discursive Writing', 4),
            Course(3010, 'Modern English Grammar', 3),
            Course(4725, 'Advanced Workshop in Poetry', 5));
BEGIN
     UPDATE department SET courses = new_courses WHERE name = 'English';
```

Oracle: пример использования вложенных таблиц

Пример добавления курса для History Department:

```
BEGIN
    INSERT INTO
        THE(SELECT courses FROM department
            WHERE name = 'History')
        VALUES(3340, 'Modern China', 4);
```

Оператор THE является подзапросом, который возвращает единственное значение соответствующего столбца courses. Этот столбец должен быть типом nested table, в противном случае возникнет ошибка выполнения.

Пример изменения размера кредита для двух курсов Psychology Department:

```
DECLARE
    adjustment INTEGER  DEFAULT  1;
BEGIN
    UPDATE     THE(SELECT courses FROM department
                   WHERE name = 'Psychology')
   SET credits = credits + adjustment
   WHERE course_no IN (2200, 3540);
```

Oracle: пример использования вложенных таблиц

Пример выбора номера и названия курса в локальные переменные:

```
DECLARE
    my_course_no NUMBER(4);
    my_title     VARCHAR2(35);
BEGIN
    SELECT course_no, title INTO my_course_no, my_title
     FROM THE(SELECT courses FROM department
                  WHERE name = 'History')
    WHERE course_no = 3105;
```

Пример удаления всех курсов с кредитом=5 для English Department:

```
BEGIN
    DELETE THE(SELECT courses FROM department
                 WHERE name = 'English')
    WHERE credits = 5;
```

Oracle: пример использования вложенных таблиц

SELECT * FROM department ; -- выбор данных

NAME DIRECTOR OFFICE

---------- ---------- ----------

COURSES(COURSE_NO, TITLE, CREDITS)

Psychology Irene Friedman Fulton Hall 133

COURSELIST(COURSE(1000, 'General Psychology', 5), COURSE(2100, 'Experimental Psychology', 4), COURSE(2200, 'Psychological Tests', 4), COURSE(2250, 'Behavior Modification', 4), COURSE(3540, 'Groups and Organizations', 4), COURSE(3552, 'Human Factors in the Workplace', 4), COURSE(4210, 'Theories of Le

arning', 4), COURSE(4320, 'Cognitive Processes', 4), COURSE(4410, 'Abnormal Psychology', 4))

History John Whalen Applegate Hall 142

COURSELIST(COURSE(1011, 'History of Europe I', 4), COURSE(1012, 'History of Europe II', 4), COURSE(1202, 'American History', 5), COURSE(2130, 'The Renaissance', 3), COURSE(2132, 'The Reformation', 3), COURSE(3105, 'History of Ancient Greece', 4), COURSE(3321, 'Early Japan', 4), COURSE(3601, 'Latin A

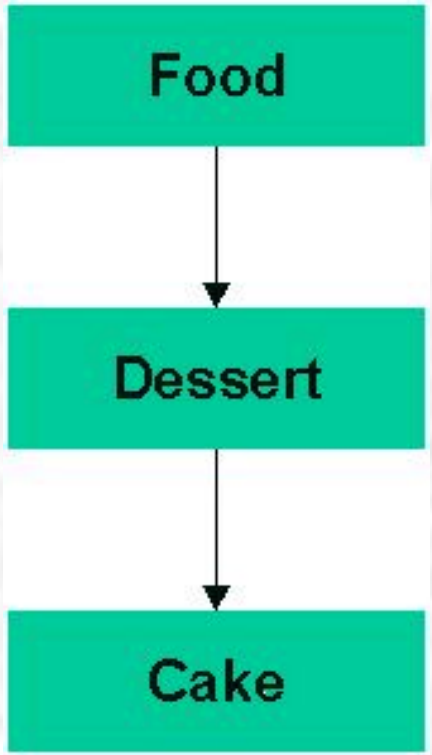
merica Since 1825', 4), COURSE(3702, 'Medieval Islamic History', 4), COURSE(3340, 'Modern China', 4))

2 rows selected.

Подставляемость и преобразование объектных типов в иерархии

Пример: иерархия ЕДА -> ДЕСЕРТ -> ПИРОЖНЫЕ

```
CREATE TYPE food_t AS OBJECT (
    name VARCHAR2(100),
    food_group                    VARCHAR2 (100),
    grown_in                      VARCHAR2 (100)
    )
    NOT FINAL
    ;
/
CREATE TYPE dessert_t UNDER food_t (
    contains_chocolate     CHAR(1),
    year_created           NUMBER(4)
    )
    NOT FINAL
    ;
/
CREATE TYPE cake_t UNDER dessert_t (
    diameter       NUMBER,
    inscription    VARCHAR2(200)
    )
    ;
/
```



Продолжение примера

```
DECLARE
    my_favorite   cake_t
    := cake_t (
          'Marzepan Delight',
          'CARBOHYDRATE',
          'Swedish Bakery',
          'N',
          1634,
          8,
          'Happy Birthday!'
       );
 BEGIN
    DBMS_OUTPUT.put_line (my_favorite.NAME);
    DBMS_OUTPUT.put_line (my_favorite.inscription);
 END;
 /
```

Примеры работы с данными этой иерархии:

- выбирать и просматривать все типы по всей иерархии;
- обновлять определенный уровень иерархии, такой как пирожные;
- работать со всеми десертами, которые не являются пирожными.

Продолжение примера

Создадим таблицу на основе ранее созданного типа:

```
CREATE TABLE sustenance OF food_t;
```

И добавим в неё данные:

```
BEGIN
    INSERT INTO sustenance
         VALUES (food_t (
                    'Brussel Sprouts',
                    'VEGETABLE',
                    'farm'
                )
                );
    INSERT INTO sustenance
         VALUES (dessert_t (
                    'Jello',
                    'PROTEIN',
                    'bowl',
                    'N',
                    1887
                )
                );
    INSERT INTO sustenance
         VALUES (cake_t (
                    'Marzepan Delight',
                    'CARBOHYDRATE',
                    'bakery',
                    'N',
                    1634,
                    8,
                    'Happy Birthday!'
                )
                );
 END;
 /
```

Продолжение примера

Выведем данные из таблицы:

```
SQL> SELECT * FROM sustenance;
 NAME                      FOOD_GROUP          GROWN_IN
 ------------------------- ------------------- ----------
 Brussel Sprouts           VEGETABLE           farm
 Jello                     PROTEIN             bowl
 Marzepan Delight          CARBOHYDRATE        bakery
```

Введем в таблицу данные о десерте и выведем их на экран:

```
DECLARE
    mmm_good food_t :=
       dessert_t (
          'Super Brownie',
          'CARBOHYDRATE',
          'my oven', 'Y', 1994);
 BEGIN
    DBMS_OUTPUT.PUT_LINE (
       mmm_good.name);
 END;
 /
```

Продолжение примера

Пример подставляемости в PL/SQL коллекциях:

```
DECLARE
    TYPE foodstuffs_nt IS TABLE OF food_t;
    fridge_contents foodstuffs_nt := (
       food_t (
         'Eggs benedict', 'PROTEIN', 'Farm'),
       dessert_t (
         'Strawberries and cream',
         'FRUIT', 'Backyard', 'N', 2001),
       cake_t (
         'Chocolate Supreme', 'CARBOHYDRATE',
         'Kitchen', 'Y', 2001,
          8, 'Happy Birthday, Veva'));
BEGIN
    FOR indx IN
       fridge_contents.FIRST ..
fridge_contents.LAST
    LOOP
       DBMS_OUTPUT.PUT_LINE (
          fridge_contents(indx).name);
    END LOOP;
END;
/
```

Продолжение примера

Как отключить подставляемость:

```
CREATE TABLE meal (
    served_on DATE,
    appetizer food_t,
    main_course food_t,
    dessert dessert_t
    )
    COLUMN appetizer NOT SUBSTITUTABLE AT ALL LEVELS
    ;
```

Предложение NOT SUBSTITUTABLE используется, чтобы указать, что при задании значения для столбца appetizer нельзя использовать подтип еды. Если попытаться вставить десерт в качестве закуски, то результатом выполнения будет следующая ошибка:

```
ERROR at line 1:
 ORA-00932: inconsistent datatypes
```

Можно применять это предложение целиком к таблице:

CREATE TABLE brunches OF food_t NOT SUBSTITUTABLE AT ALL LEVELS;

Продолжение примера

Расширение и сужение объектных типов

Расширение - это присвоение, в котором объявленный тип источника является более конкретным, чем объявленный тип места назначения. Если присваивается объект (или экземпляр объектного типа, если говорить точнее) типа cake_t переменной типа dessert_t, то выполняется расширение.

Сужение - это присвоение, в котором объявленный тип источника является более общим, чем объявленный тип места назначения. Если присваивается объект типа dessert_t переменной типа cake_t, то выполняется операция сужения.

Расширение является фактически "родным" для иерархий объектных типов Oracle и их свойства подставляемости.

Продолжение примера

Сужение:
Функция **TREAT** явно изменяет объявленный тип источника в присваивании на более специализированный тип или подтип в иерархии места назначения:

TREAT (<object instance> AS <object type>)

<object instance> - это значение столбца или строки коллекции данного конкретного супертипа в объектной иерархии,
<object type> - это подтип в этой иерархии.

Выбрать все обеды, в которых в качестве основного блюда указан десерт:

```
SELECT TREAT (main_course AS dessert_t).contains_chocolate chocolatey
    FROM meal
    WHERE TREAT (main_course AS dessert_t) IS NOT NULL;
```