Введение в Oracle, создание баз данных под управлением СУБД Oracle, типы данных

1. История СУБД Oracle
2. Архитектура СУБД Oracle
3. Версии СУБД Oracle
4. Утилиты:
    a. SQL Plus
    b. Database Configuration Assistant
    c. Administration Assistant for Windows
    d. Net Configuration Assistant
5. Инсталляция СУБД Oracle.
6. Архитектура БД под управлением Oracle. Сравнения с базами данных других СУБД.
7. Создание базы данных с помощью Database Configuration Assistant
8. Создание базы данных с помощью файла конфигурации
9. Обзор типов данных СУБД Oracle.
10. Создание структуры учебной базы данных с использованием СУБД Oracle.

1. История СУБД Oracle

В история СУБД ORACLE можно выделить несколько основных этапов:

1. Разработка проекта Oracle для нужнд ЦРУ.
2. Появление компании Software Development Lab., создание ее первой системы Oracle 2, вдохновителем для которой явился проект System R.
3. Решение проблемы переносимости ORACLE за счет рализация Oracle 3 на языке C.
4. Повышение надежности СУБД ORACLE, выход версии Oracle 4, портирование Oracle на персональный компьютер.
5. Реализация архитектуры Client/Server в версии Oracle 5.
6. Обработка транзакций в режиме реального времени в Oracle 6.
7. Расширение языка SQL средствами описания процедур и триггеров БД с появлением Oracle 7. Появление разделяемого пула процессов сервера для взаимодействия с клиентами.
8. Ориентирование на интернет-технолоии начиная с версии Oracle 8i (символ "i" в названии новой верисии говорит о направленности на Internet).

Более подробно об истории развития СУБД ORACLE можно прочитать статью "Компьютерные корпорации. История Oracle." на сайте Компьютерной газеты A-Z (http://www.nestor.minsk.by/kg/2005/26/kg52614.html).

На сегодняшя, с момента появления Интернета самой передовой технологией является Oracle Enterprise Grid, а также Oracle10g Database. В странах СНГ Oracle делает ставку на предоставление консалтинга.

2. Архитектура СУБД Oracle

Oracle проектировалась как переносимая СУБД. Физическая архитектура Oracle различна в разных операционных системах. В ОС UNIX Oracle представлена в виде отдельных процессов, где каждая функция СУБД представляет собой отдельный процесс. В ОС Windows Oracle реализована как один многопоточный процесс. В ОС Netware тоже используется многопоточная модель. Хотя физические средства реализации СУБД Oracle на разных платформах отличаются, но архитектура системы общая.

Существуют 3 основных компонента Oracle.

- **Файлы**. Файлы параметров, сообщений, данных, временных данных и журналов повторного выполнения.
- **Структуры памяти**, System Global Area (SGA).
- **Потоки (процессы)**. Существуют серверные, фоновые и подчиненные потоки.

Сервер

Потоки Oracle тесно связаны с компонентами SGA. Ниже даны некоторые определения необходимые для дальнейшего обзора сервера Oracle.

Под термином "экземпляр" понимаются процессы и память сервера. «БД» предполагает физические файлы данных. БД доступна многим экземплярам, но не наоборот. Чаще всего между БД и экземпляром задано отношение один к одному.

Oracle включает область памяти SGA, содержащую структуры данных, для кеширования дисковых операций, хранения планов выполнения разобранных операторов SQL и т.д. В Oracle имеется файлы, читаемые и записываемые только потоками БД. В этих файлах хранятся данные, индексы, журналы повторного выполнения и т.д.

Фоновые потоки Oracle — потоки, образующие экземпляр; появляются при запуске СУБД. В Windows утилита **tlist (**resource toolkit) обнаруживает только один процесс — **Oracle.exe**. Данный процесс имеет несколько потоков, представляющих фоновые потоки Oracle:

```
...\Desktop>tlist 1072

1072 ORACLE.EXE

   CWD:     C:\oracle\DATABASE\

   CmdLine: c:\oracle\bin\ORACLE.EXE TKYTE816

   VirtualSize:   144780 KB   PeakVirtualSize:   154616 KB

   WorkingSetSize: 69424 KB   PeakWorkingSetSize: 71208 KB
```

```
  NumberOfThreads: 11
     0 Win32StartAddr:0x00000000 LastErr:0x00000000 State:Initialized
     5 Win32StartAddr:0x00000000 LastErr:0x00000000 State:Initialized
     5 Win32StartAddr:0x00000000 LastErr:0x00000000 State:Initialized
     5 Win32StartAddr:0x00000000 LastErr:0x00000000 State:Initialized
     5 Win32StartAddr:0x00000000 LastErr:0x00000000 State:Initialized
     5 Win32StartAddr:0x00000000 LastErr:0x00000000 State:Initialized
     5 Win32StartAddr:0x00000000 LastErr:0x00000000 State:Initialized
     5 Win32StartAddr:0x00000000 LastErr:0x00000000 State:Initialized
     5 Win32StartAddr:0x00000000 LastErr:0x00000000 State:Initialized
     5 Win32StartAddr:0x00000000 LastErr:0x00000000 State:Initialized
     5 Win32StartAddr:0x00000000 LastErr:0x00000000 State:Initialized
       0.0.0.0 shp  0x00400000  ORACLE.EXE
    5.0.2163.1 shp  0x77f80000  ntdll.dll
       0.0.0.0 shp  0x60400000  oraclient8.dll
       0.0.0.0 shp  0x60600000  oracore8.dll
       0.0.0.0 shp  0x60800000  oranls8.dll
...
```

Утилита отобразила 11 потоков Oracle. При подключении к БД, количество потоков увеличится до 12.

При подключении пользователя Oracle создает новый процесс, на все время сеанса ему выделяется отдельный серверный процесс. Сеансы и выделенные серверы находятся в отношении один к одному. Клиент, т.е. любая программа, соединяющаяся с БД будет взаимодействовать с сервером по сети. Этот сервер будет получать и выполнять SQL-операторы, в этом заключается его прямое назначение.

СУБД Oracle может работать в режиме multi-threaded server (MTS), в котором при подключении нового клиента не создается дополнительный поток. В режиме MTS СУБД Oracle создает пул "разделяемых серверов" для поддержки большого количества пользователей. В данном режиме серверные потоки запускаются сразу при старте СУБД.

Рассмотрим пример подключения утилиты SQL*Plus по сети по протоколу TCP/IP. Клиент и сервер могут находятся на разных компьютерах, связанных сетью по протоколу TCP/IP или на одном компьютере между основной ОС (клиент) и виртуальной машиной (сервер). Клиент посылает запрос на подключение к БД:

```
C:\> sqlplus scott/tiger@ora816.us.oracle.com
```

scott – логин, **tiger** — пароль, а **ora816.us.oracle.com** — имя службы TNS. Transparent Network Substrate (TNS) — прозрачная сетевая среда, программное обеспечение, интегрированное в любое клиентское приложение Oracle, которое обеспечивает удаленное взаимодействие клиента и сервера. Строка подключения задает параметры подключения к БД. Клиент использует файл **TNSNAMES.ORA**, благодаря которому строка **ora816.us.oracle.com** трансформируется в параметры «хост», «порт» и идентификатор БД (SID). Это текстовый файл конфигурации, находящийся в каталоге **[ORACLE_HOME]\network\admin**:

```
ORA816.US.ORACLE.COM =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = aria.us.oracle.com)(PORT = 1521))
    )
    (CONNECT_DATA =
      (ORACLE_SID = ora816)
    )
  )
```

Если служба сервера Net8 настроена и запущен процесс-слушатель (TNS Listener), то соединение может установлено.

3. Полный перечень версий ORACLE можно увидеть по следующей ссылке:

http://ru.wikipedia.org/wiki/Oracle_%28%D0%A1%D0%A3%D0%91%D0%94%29

4. а. Работа с утилитой SQL*Plus

Выполняем подключение к Oracle:

```
SQL> CONNECT SYSTEM/MANAGER@[имя_сетевой_службы];
```

Вводим первую команду:

```
SELECT COUNT(*) FROM USER_OBJECTS
```

Нажмите **Enter** и увидите:

```
SQL> SELECT COUNT(*) FROM USER_OBJECTS
  2
```

Это интерпретатор команд предлагает ввести следующую строку для вашего выражения если написать, что-то еще, то появится:

```
  3
  4
и т.д.
```

То есть среда предлагает ввести выражение построчно в виде произвольного количества строк. Например, если Вы намереваетесь, записать какой либо сложный запрос…

Чтобы увидеть результат введите символ "/" и снова нажмите **Enter**. Это означает, что вы закончили ввод и хотите получить результат.

Должно получиться примерно следующее:

```
 COUNT(*)
---------
       64
```

Это значит, что, пользователь **SYSTEM**, имеет в своей схеме 64 объекта БД (в зависимости от версии Oracle могут получаться произвольные значения).

В ответ на приглашение **SQL>** введите еще раз "/" и **Enter**. Получаем аналогичный результат:

```
 COUNT(*)
---------
       64
```

Это значит, что наше выражение «где-то внутри» и поле ввода "/" может выполняться произвольное количество раз! Выражение "/" можно заменить, написав **RUN** следующим образом:

```
SQL> RUN
  1*  SELECT COUNT(*) FROM USER_OBJECTS

 COUNT(*)
---------
       64
```

Теперь мы снова увидели выражение, которое содержится в строковом буфере.

Если же необходимо посмотреть весь строковый буфер **SQL Plus**, то вводим следующее:

```
SQL> LIST
```

Получаем:

```
SQL> LIST
  1*  SELECT COUNT(*) FROM USER_OBJECTS
```

Еще она полезная команда **SQL Plus** это **EDIT**. Вводим:

```
SQL> EDIT
```

После нажатия **Enter** появится Блокнот, как средство для написания скриптов **SQL**! В окне блокнота, будет строка:

```
SELECT COUNT(*) FROM USER_OBJECTS
/
```

Если ничего не менять, то она же и останется, если вместо нее написать, например:

```
SELECT COUNT(*) FROM USER_TABLES
/
```

То при закрытии будет задан вопрос: Сохранить изменения в файле **afiedt.buf**!

Он будет содержать, то, что хранит буфер, он обычно находится в каталоге [disk]**:\Oracle\Bin**! Сохраняем, и видим следующее:

```
  1* SELECT COUNT(*) FROM USER_TABLES
SQL>
```

Вводим "/", и получаем результат:

```
 COUNT(*)
---------
       17
```

У пользователя **SYSTEM** 17 таблиц в схеме.

SQL*Plus позволяет загружать текстовые файлы в блокнот с диска. Например можно сделать следующее: создать каталог на вашем диске, скажем **Temp** и поместим в него файл содержащий следующее:

```
C:\Temp\proba.txt
Hello world!!!
```

Вводим:

```
SQL> EDIT C:\temp\proba.txt
```

Откроется Блокнот и покажет содержимое файла.

Если экран SQL Plus сильно «захламлен» – введите CLEAR SCREEN и будет выполнена очистка экрана. Для очистки буфера команд – напишите CLEAR BUFFER.

* * *

Если при выполнении SQL-запроса или программы PL/SQL обнаружены ошибки, то в первую очередь надо проверить правильность написания в них имен таблиц, столбцов и др. Для этого можно воспользоваться командой SQL*Plus DESCRIBE, которая выводит список столбцов для таблицы или спецификацию для функции, процедуры, пакета. Синтаксис этой команды имеет вид:

```
DESC[RIBE] {[user.]table [column] | [user.]object[.subobject]}
```

Пример:

```
SQL> desc kadry
 Name                              Null?    Type
 --------------------------------- -------- ----
 NOMER                             NOT NULL NUMBER(6)
 FAMILIYA                                   VARCHAR2(20)
 IMYA                                       VARCHAR2(15)
 OTCHESTVO                                  VARCHAR2(20)
 ROZHDENIE                                  DATE
 POL                                        CHAR(1)
 IZMEN                             NOT NULL DATE
```

Аналогичным образом можно получить структуру любого представления словаря данных, например, структуру user_objects, где хранится информация о пользовательских объектах базы данных *(INDEX, SEQUENCE, VIEW, PACKAGE, PACKAGE BODY, FUNCTION, PROCEDURE, TABLE, TRIGGER)*:

```
desc user_objects
 Name              Null?    Type
 ----------------- -------- ----
 OBJECT_NAME                VARCHAR2(128)
 OBJECT_ID                  NUMBER
```

```
OBJECT_TYPE                    VARCHAR2(13)

CREATED                        DATE

LAST_DDL_TIME                  DATE

TIMESTAMP                      VARCHAR2(75)

STATUS                         VARCHAR2(7)
```

Для получения полного описания всех объектов можно выполнить команду:

```
select * from user_objects;
```

а для получения описания процедур

```
select * from user_objects where object_type = 'PROCEDURE';
```

```
OBJECT_NAME OBJECT_ID OBJECT_TYPE CREATED    LAST_DDL_T TIMESTAMP           STATUS

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

PR_CURS     1928      PROCEDURE   17.11.1996 02.02.1997 1997-02-02:13:37:05 VALID

PR_PRINT    1957      PROCEDURE   24.11.1996 24.11.1996 1996-11-24:16:59:44 INVALID

PR_SHTAT    1970      PROCEDURE   02.01.1997 08.01.1997 1997-01-08:12:38:02 VALID
```

При возникновении ошибок в командах *CREATE PACKAGE, CREATE PACKAGE BODY, CREATE PROCEDURE, CREATE FUNCTION, CREATE TRIGGER, CREATE VIEW* их уточнение можно выполнить с помощью команды SQL*Plus:

```
SHOW ERR[ORS] [{PACKAGE | PACKAGE BODY | PROCEDURE | FUNCTION |TRIGGER
| VIEW} name]
```

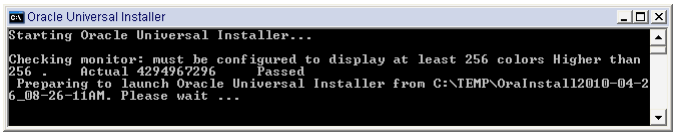
5. Инсталляция СУБД ORACLE

При описании инсталляции используется следующая версия программно обеспечения: ORACLE 11*g* Release 2 (11.2) for Microsoft Windows (32-Bit).

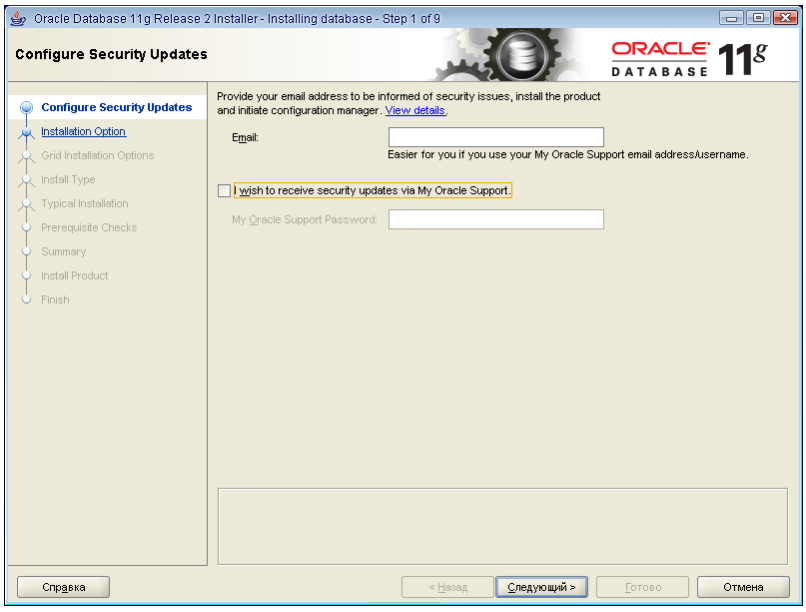
Версия доступна для скачивания на официальном сайте **ORACLE** после предварительной регистрации. Также доступна версия и для 64-разрядных платформ.

После скачивания, инсталляционный пакет представлен двумя файлами архивов в формате Zip: ***win32_11gR2_database_1of2.zip*** и ***win32_11gR2_database_2of2.zip***.

Этап подготовительный, ORACLE производит инициализацию инсталлятора:



1. Указание e-mail пользователя на который будут поступать информация от компании ORACLE об имеющихся обновлениях. В случае, когда предоставлена возможность сопровождения, во втором поле указывается пароль для службы тех-поддержки.

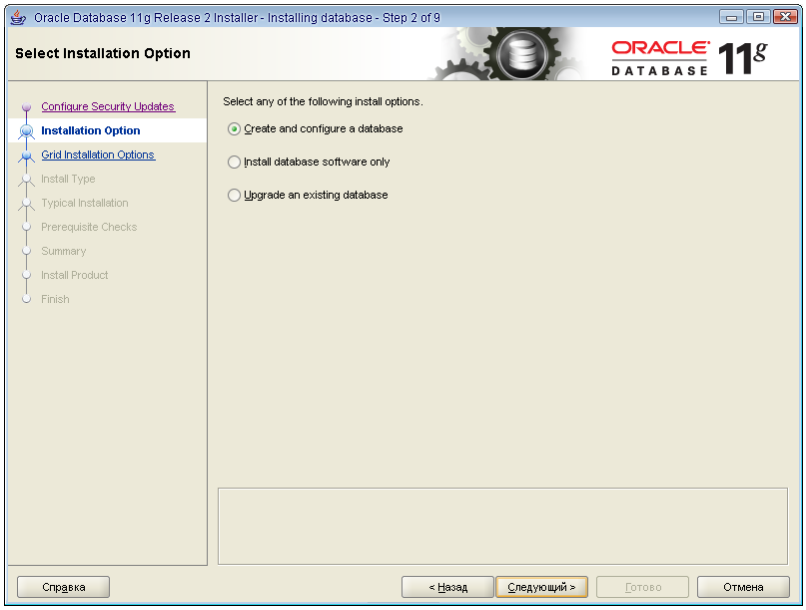


В нашем случае в первом поле можно указать свой e-mail и снять галочку.

2. Выбор вида осуществляемых действий:

- Создание и конфигурирование СУБД (по-умолчанию).
- Инсталляция СУБД без выполнения конфигурирования.
- Обновление существующей СУБД (при переходе на более новую версию СУБД)

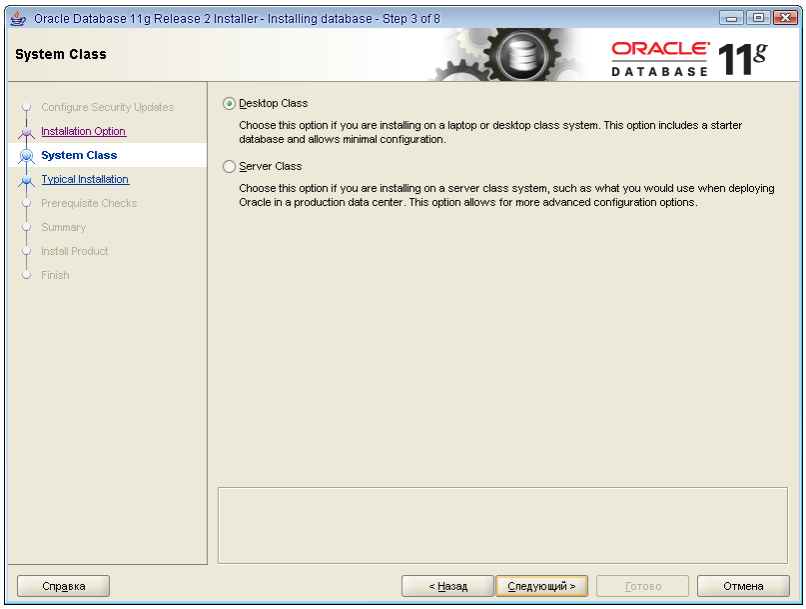
Нам подходит первый вариант, т.к. именно он позволит установить программное обеспечение и базы данных, предоставляемые ORACLE для обучения.



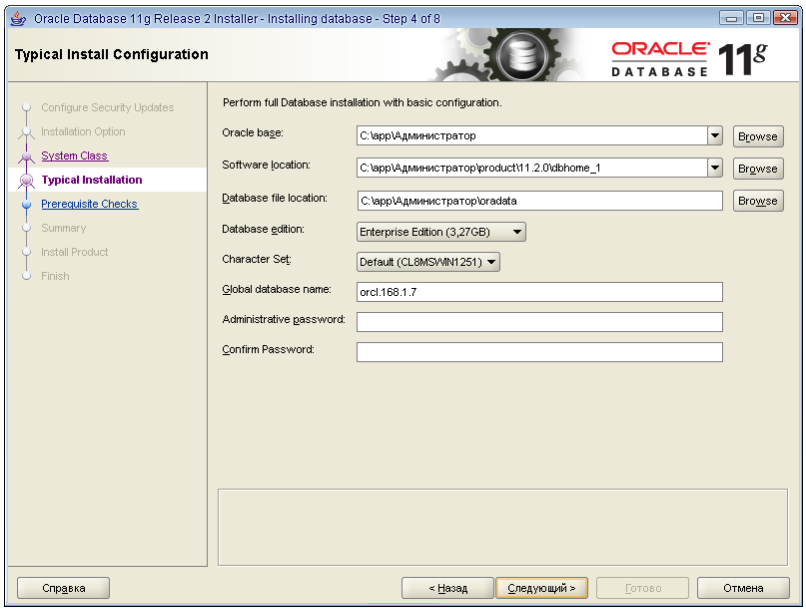
3. Выбор вида инсталляции:

- Класс рабочей станции (требует минимальных настроек).
- Класс сервера (используется при профессиональной конфигурации работы сервера).

Нам, как разработчикам БД, для учебных целей, подходит первый вариант инсталляции.


Oracle Database 11g Release 2 Installer - Installing database - Step 3 of 8
System Class
ORACLE DATABASE 11g
Configure Security Updates
Installation Option
System Class
Typical Installation
Prerequisite Checks
Summary
Install Product
Finish
Desktop Class
Choose this option if you are installing on a laptop or desktop class system. This option includes a starter database and allows minimal configuration.
Server Class
Choose this option if you are installing on a server class system, such as what you would use when deploying Oracle in a production data center. This option allows for more advanced configuration options.
Справка
< Назад
Следующий >
Готово
Отмена

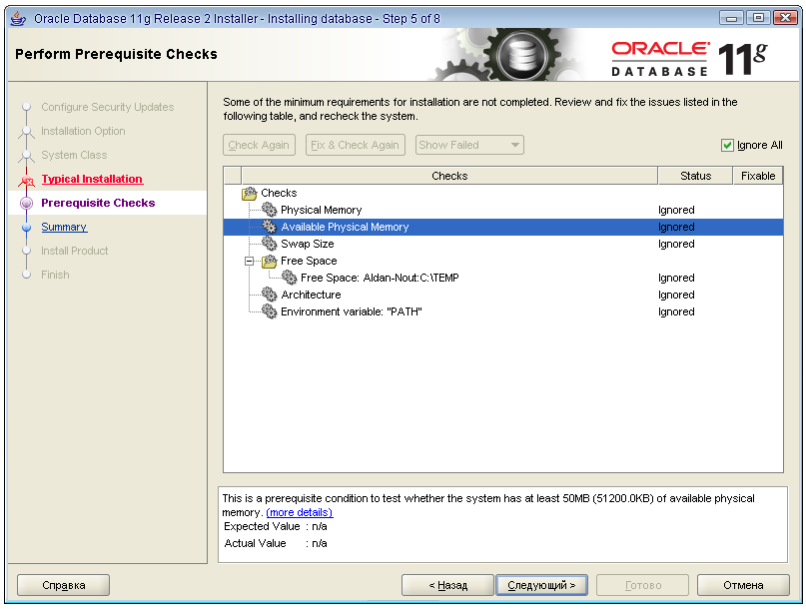
4. Конфигурирование СУБД. Выбор пути расположения файлов СУБД и фалов баз данных. Выбор вида инсталляции СУБД. Указание пароля администратора и подтверждение пароля.



СУБД ORACLE доступно в нескольких редакциях. Первый вариант, указанный по-умолчанию (Enterprice Edition), является наиболее полным и приемлемым для нас. О других редакциях можно прочитать в документации по ORACLE или на официальном сайте.

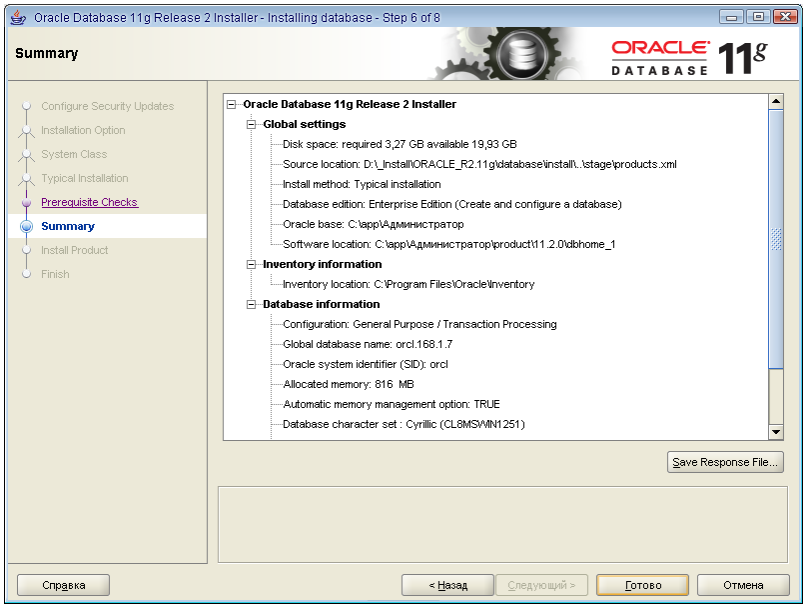
ВАЖНО!!!: Указанный Вами пароль будет неоднократно использоваться далее. Прежде, чем перейти с следующему шагу запомните указанный пароль.

5. Проверка системных требований. Системный требования для ORACLE можно посмотреть в документе, входящем в состав инсталляции: database/welcome.html -> *Database Quick Installation Guide* .



В случае, если по каким-либо причинам проверку осуществить не удается или какие-то критерии не удовлетворяют минимальным требованиям для того, чтобы пройти этот шаг, можно установить галочку «Ignore All». Это не относится к таким критичным требованиям как свободное дисковое пространство и минимальный объем оперативной памяти.

6. Суммарная информация о предыдущих результатах работы инсталлятора:

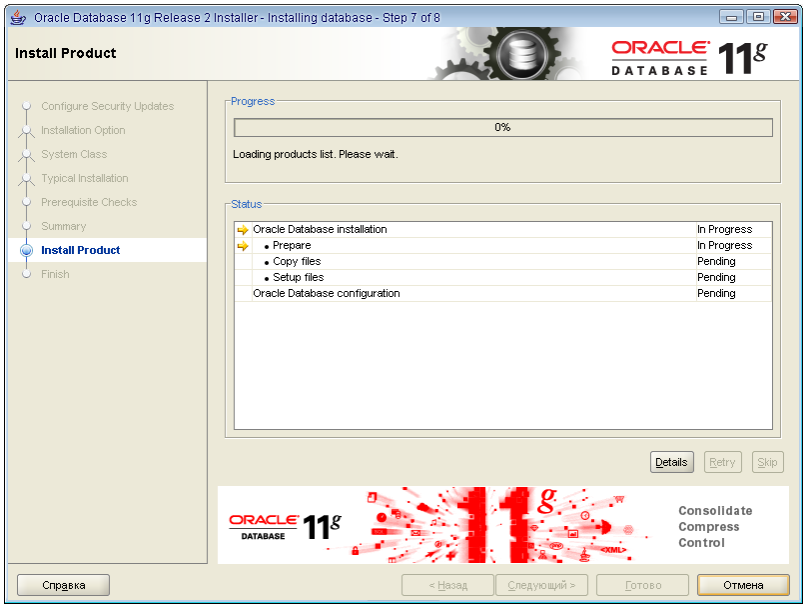


Представленная информация дает картину того, где будут располагаться файлы ORACLE после инсталляции, название идентификатора экземпляра ORACLE (SID), доступном дисковом пространстве и т.д.

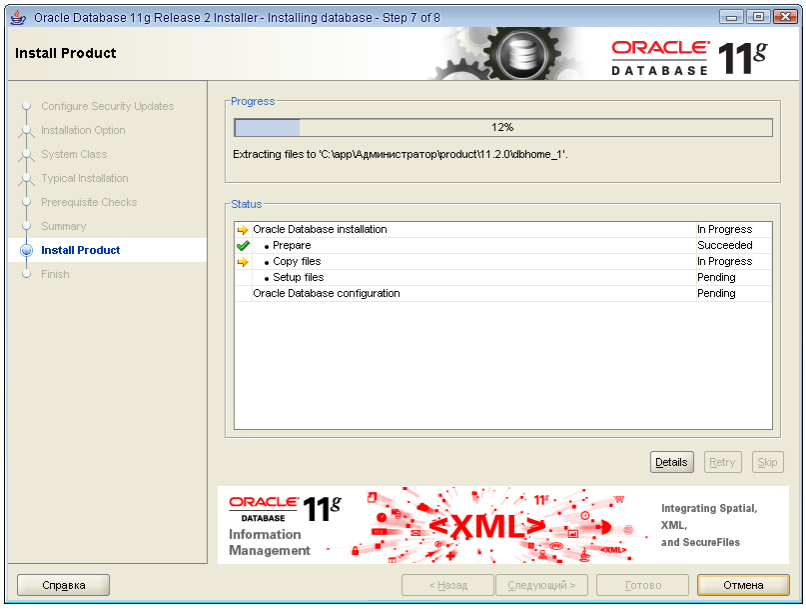
Жмем кнопку « Готово ».

7. Процесс установки файлов и конфигурирования СУБД.

Подготовка к инсталляции:



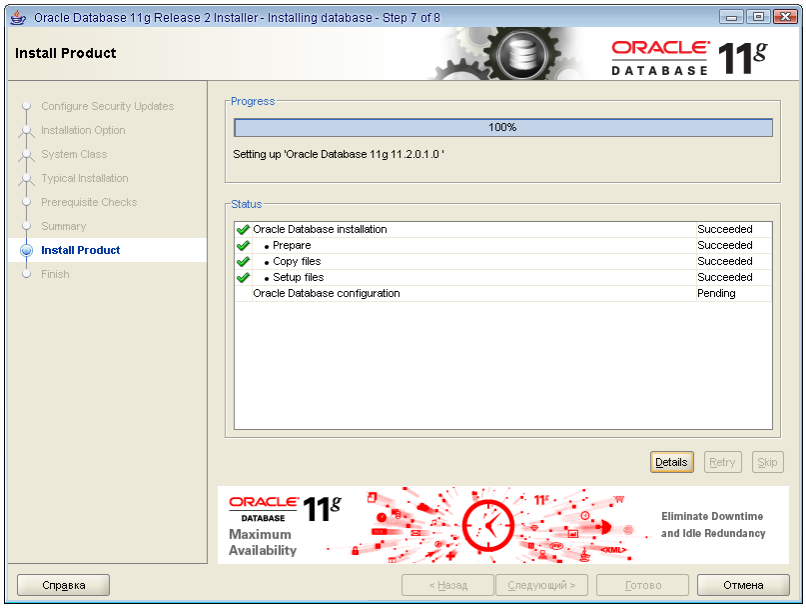
Копирование файлов:



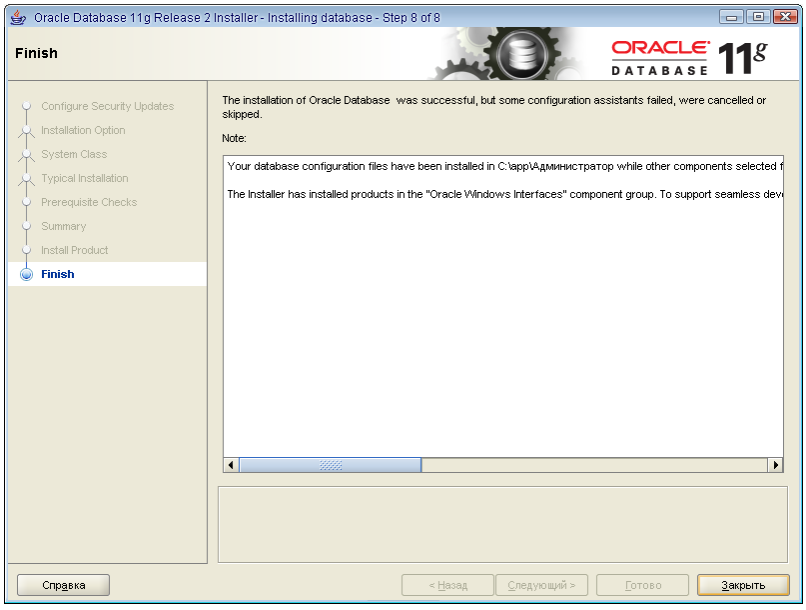
При нажатии кнопки « Details » можно посмотреть детальную информацию о ходе инсталляции:


Details
Loading products list. Please wait.
Loading products. Please wait.
Analyzing dependencies
Setting up install types
Executing pre-requisites
Loading Oracle Database 11g
Loading Enterprise Edition Options
Loading Oracle Partitioning
Loading Oracle Spatial
Loading Oracle OLAP
Loading Oracle COM Automation Feature
Loading Oracle Database 11g
Loading Oracle Text
Loading Oracle Net Services
Loading Oracle Enterprise Manager Console DB
Loading HAS Files for DB
Loading Enterprise Manager Agent
Loading Oracle Advanced Security
Loading Oracle JVM
Loading Oracle Windows Interfaces
Loading Database Configuration and Upgrade Assistants
Loading Oracle Net Listener
Loading Oracle Internet Directory Client
Loading Oracle XML Development Kit
Loading Oracle Multimedia
Loading Oracle Call Interface (OCI)
Loading Oracle Multimedia Locator
Loading SQL*Plus
Loading Oracle Database Utilities
Loading Oracle Programmer
Loading Precompiler Common Files
Loading Oracle Database Gateway for ODBC
Loading Generic Connectivity Common Files
Loading Oracle Recovery Manager
Loading Oracle Net
Loading Oracle Netca Client
Loading Oracle LDAP administration
Loading Oracle Help for the Web
Loading Assistant Common Files
Log File: C:\Program Files\Oracle\Inventory\logs\installActions2010-04-26_08-36-42AM.log
Close

Настройка файлов СУБД и системных файлов:



Окончание установки:



6. Архитектура БД под управлением ORACLE. Сравнения с базами данных других СУБД.

База данных ORACLE имеет как физическую, так и логическую структуру. За счет разделения физической и логической структуры базы данных достигается возможность управления физической структурой данных, не затрагивая доступа к логическим структурам данных.

Физическая структура базы данных ORACLE определяется файлами операционной системы, из которых состоит база данных. Каждая база данных ORACLE составляется из файлов трех типов: одного или нескольких файлов данных, двух или более файлов журнала повторения работы и одного или нескольких управляющих файлов. Файлы базы данных предоставляют действительную физическую память для информации базы данных.

Логическая структура базы данных ORACLE определяется:

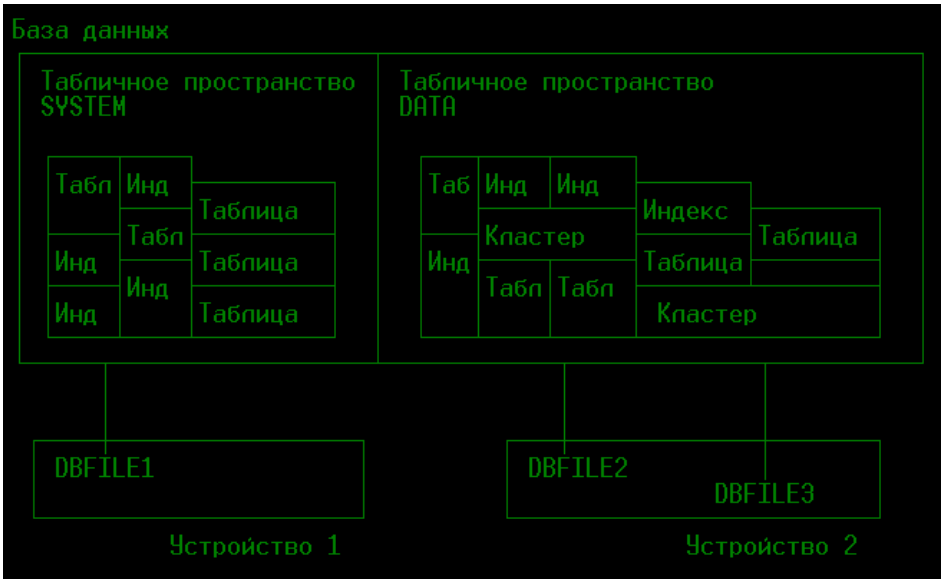
- одним или несколькими табличными пространствами
- объектами схем базы данных (таблицами, обзорами, индексами, кластерами, последовательностями, хранимыми процедурами)

Логические структуры хранения, включая табличные пространства, сегменты и экстенты, определяют, как используется физическое пространство базы данных. Объекты схем и отношения между ними формируют реляционную структуру базы данных.

Схемы и объекты схем

С каждым пользователем базы данных ассоциируется СХЕМА. Схема представляет собой коллекцию объектов схемы. Примерами объектов схемы могут служить таблицы, обзоры, последовательности, синонимы, индексы, кластеры, связи баз данных, процедуры и пакеты.

Объекты схемы – это логические структуры памяти данных. Каждому объекту схемы не соответствует физический файл на диске, в котором содержится информация этого объекта. Однако объект схемы логически хранится в табличном пространстве базы данных. Данные каждого объекта физически размещаются в одном или нескольких файлах данных табличного пространства. Для некоторых объектов, таких как таблицы, индексы и кластеры, вы можете указать количество дисковой памяти, которое должно быть распределено объекту в файлах данных табличного пространства. Рисунок показывает отношение между объектами, табличными пространствами и файлами данных:



Не существует отношения между схемами и табличными пространствами; табличные пространства могут содержать объекты из разных схем, а объекты одной схемы могут содержаться в разных табличных пространствах.

Таблицы

ТАБЛИЦА – это основная единица памяти данных в базе данных ORACLE. Данные таблицы хранятся в СТРОКАХ и СТОЛБЦАХ. Каждая таблица определяется с ИМЕНЕМ (таким как EMP) и набором столбцов. Каждому столбцу дается ИМЯ СТОЛБЦА (такое как EMPNO, ENAME или JOB), ТИП ДАННЫХ (такой как VARCHAR2, DATE или NUMBER) и ШИРИНА (которая может быть предопределена типом данных, как в случае DATE) или ТОЧНОСТЬ и МАСШТАБ (только для столбцов типа NUMBER). Строка – это коллекция информации столбцов, соответствующая одной записи.

Вы можете специфицировать правила для каждого столбца таблицы. Эти правила называются ОГРАНИЧЕНИЯМИ ЦЕЛОСТНОСТИ. Примером может служить ограничение целостности NOT NULL, которое диктует, что столбец не может содержать пустых значений.

После того как вы создали таблицу, в нее можно вставлять строки данных, используя предложения SQL. Затем информацию таблицы можно опрашивать, удалять или обновлять с помощью SQL.

Хранение данных таблицы

При создании некластеризованной таблицы в табличном пространстве автоматически создается сегмент данных, в котором будут храниться будущие данные этой таблицы. Вы можете контролировать распределение и использование пространства для сегмента данных таблицы различными способами:

- Можно управлять размерами экстентов для сегмента данных, установив параметры пространства для этого сегмента.
- Можно контролировать использование свободной памяти в блоках данных, составляющих экстенты сегмента данных, установив параметры PCTFREE и PCTUSED для этого сегмента.

Каждая строка таблицы базы данных хранится как один или несколько кусков строки. Если вся строка может быть вставлена в один блок данных, она первоначально вставляется как единственный кусок. Однако, если все данные в строке не могут быть вставлены в один блок данных, или если в результате обновления строки ее данные переполняют ее блок данных, то строка сохраняется как несколько кусков. Блок данных обычно содержит лишь один кусок для каждой строки данных; все данные строки, которые умещаются в блоке, хранятся в одном куске строки внутри блока. Когда требуется разбить строку на несколько кусков, образуется "цепочка" из кусков этой строки, размещенных в нескольких блоках данных. Куски строки логически сцепляются через идентификаторы строки (ROWID'ы).

Каждый кусок строки, в цепочке он или нет, содержит ЗАГОЛОВОК СТРОКИ и данные для всех или некоторых столбцов строки. Индивидуальные значения столбцов также могут занимать несколько кусков строки и, следовательно, блоков данных.

ЗАГОЛОВОК СТРОКИ предшествует данным строки и содержит следующую информацию:

- информацию об этом куске строки
- информацию о цепочке, если этот кусок в цепочке
- информацию о столбцах в этом куске строки
- для кластеризованных данных - информацию ключа кластера

Некластеризованная строка, полностью умещающаяся в одном блоке, имеет не менее чем трехбайтовый заголовок строки. Вслед за информацией заголовка строки каждая строка содержит информацию о столбцах и данные столбцов. На длину столбца требуется один байт, если эта длина не превышает 250 байт, или три байта для столбцов, содержащих более 250 байт; это поле длины столбца непосредственно предшествует данным столбца. Память, занимаемая данными столбца, зависит от типа данных столбца. Если столбец имеет переменную длину, то место, занимаемое значением столбца, может уменьшаться и увеличиваться при обновлениях.

Для экономии места, строка, содержащая только пустые значения, хранится только как нулевая длина столбца, без данных.

Замечание: На каждую строку используется два байта в оглавлении строк в заголовке блока данных.

Порядок столбцов

Порядок столбцов один и тот же для всех строк в данной таблице. Столбцы обычно хранятся в том порядке, в котором они были перечислены в предложении CREATE TABLE, но это не гарантируется. Например, если вы создаете таблицу со столбцом типа LONG, то ORACLE

всегда размещает этот столбец последним. Кроме того, если таблица изменяется так, что к ней добавляется новый столбец, то этот новый столбец становится последним хранящимся столбцом.

В общем случае, вы должны стараться последними размещать те столбцы, которые часто содержат пустые значения, с тем, чтобы строки занимали меньше места. Заметьте, однако, что, если ваша таблица содержит столбец типа LONG, то преимущества от перемещения таких пустых столбцов теряются.

ROWID'ы кусков строк

Каждый кусок строки идентифицируется ее адресом, который называется ROWID. Однажды назначенный ROWID сохраняется за всеми кусками данной строки до тех пор, пока строка не будет удалена, или экспортирована и импортирована с помощью утилит EXPORT и IMPORT. Поскольку ROWID является постоянным на все время жизни (куска) строки, к нему можно обращаться в предложениях SQL, и он может оказаться полезным в предложениях SELECT, UPDATE и DELETE.

Пустые значения

NULL, или пустое значение, обозначает отсутствие значения в столбце строки. Пустые значения указывают на то, что данные отсутствуют, неизвестны или неприменимы. Пустое значение не может подразумевать никакого значения, в том числе нулевого. Столбец может содержать пустые значения, если для него не было определено ограничение целостности NOT NULL; в этом случае в таблицу нельзя вставить строку, которая содержала бы пустое значение для данного столбца.

Пустые значения хранятся в базе данных, если они попадают между столбцами, имеющими значения. В таких случаях пустое значение занимает один байт. Хвостовые пустые значения не хранятся и не занимают памяти. В таблицах, содержащих много столбцов, те столбцы, которые чаще всего будут пустыми, следует определять в конце, чтобы сэкономить дисковую память.

Пустые значения распознаются в SQL через предикат IS NULL. Функция SQL с именем NVL может использоваться, чтобы преобразовывать пустые значения в непустые.

Пустые значения не индексируются, исключая случай, когда столбец ключа кластера имеет пустое значение.

Умалчиваемые значения столбцов

Столбцу таблицы может быть назначено умалчиваемое значение, так что при вставке в таблицу новой строки, если значение этого столбца опущено, ему будет автоматически присваиваться значение по умолчанию. Умалчиваемые значения столбцов работают так, как если бы они были явно специфицированы в предложении INSERT.

Если для столбца явно не определено умалчиваемое значение, то умолчанием для столбца неявно становится пустое значение (NULL).

Обзоры

ОБЗОР – это настроенное по заказу представление данных из одной или нескольких таблиц. Обзор можно рассматривать как «хранимый запрос».

Обзоры в действительности не содержат данных; вместо этого они доставляют данные из тех таблиц, на которых они основаны (так называемых БАЗОВЫХ ТАБЛИЦ обзоров). Базовые таблицы, в свою очередь, могут быть как таблицами, так и обзорами.

Как и таблицы, обзоры можно опрашивать, обновлять и осуществлять в них вставки и удаления, с некоторыми ограничениями. Все операции, осуществляемые над обзором, в действительности затрагивают базовые таблицы этого обзора.

Широкое применение обзоров обусловлено следующими их свойствами:

- Обзоры предоставляют дополнительный уровень защиты таблиц, ограничивая доступ предопределенным множеством строк и столбцов базовой таблицы. Например, обзор можно составить так, что столбцы со специфической информацией (скажем, сведениями о зарплате) не включаются в определение обзора.
- Обзоры позволяют скрыть сложность данных. Например, единственный обзор может служить для построения СОЕДИНЕНИЯ, которое является отображением взаимосвязанных столбцов или строк из нескольких таблиц. Однако такой обзор скрывает тот факт, что эти данные на самом деле принадлежат разным таблицам.
- Обзоры помогают упростить команды для пользователя. Например, с помощью обзора пользователь может выбирать информацию из нескольких таблиц, не зная, как осуществлять сложный коррелированный запрос.
- Обзоры представляют данные с иной точки зрения, чем они представлены в таблице. Например, с помощью обзора можно переименовать столбцы, не затрагивая самих таблиц, на которых базируется обзор.
- Обзоры позволяют составлять и сохранять сложные запросы. Например, запрос может выполнять обширные вычисления по информации таблицы. Благодаря тому, что этот запрос сохраняется как обзор, вы выполняете эти вычисления только при обращении к обзору.

Целостность данных

Весьма важно гарантировать подчиненность данных некоторым организационным правилам, которые определяются администратором базы данных или разработчиком приложений. Если предложение INSERT или UPDATE пытается нарушить это правило целостности, ORACLE должен откатить некорректное предложение и возвратить приложению ошибку. Для решения проблем, связанных с соблюдением правил целостности данных, ORACLE предлагает такие средства, как ограничения целостности и триггеры базы данных.

Ограничения целостности

ОГРАНИЧЕНИЕ ЦЕЛОСТНОСТИ – это декларативный способ задания организационного правила для столбца таблицы. Ограничение целостности представляет собой утверждение о данных таблицы, которое должно быть всегда истинным:

- Если для таблицы создается ограничение целостности, и в таблице уже существуют данные, не удовлетворяющие этому ограничению, то такое ограничение нельзя ввести в действие.
- После того как ограничение определено, если любые результаты предложения DML нарушают это ограничение, предложение откатывается, и возвращается ошибка.

Ограничения целостности определяются с таблицей и сохраняются как часть определения таблицы, централизованно в словаре данных базы данных, так что все приложения базы данных должны подчиняться одному и тому же набору правил. Если правило изменяется, его требуется изменить лишь один раз, на уровне базы данных, а не много раз для каждого приложения.

ORACLE поддерживает следующие ограничения целостности:

NOT NULL	запрещает пустые значения (NULL) в столбце таблицы
UNIQUE	запрещает повторяющиеся значения в столбце или группе столбцов
PRIMARY KEY	(первичный ключ) запрещает повторяющиеся и пустые значения в столбце или группе столбцов
FOREIGN KEY	(внешний ключ) требует, чтобы каждое значение в столбце или группе столбцов совпадало со значением столбца UNIQUE или PRIMARY KEY другой таблицы
CHECK	Запрещает значения, которые не удовлетворяют логическому выражению ограничения

Ключи

Термин "ключ" используется в определениях различных типов ограничений целостности. КЛЮЧ – это столбец или группа столбцов, участвующих в определении некоторых типов ограничений целостности. Ключи описывают отношения между различными таблицами и столбцами реляционной базе данных. Существуют следующие типы ключей:

Первичный ключ	Столбец или группа столбцов, включенных в определение ограничения таблицы PRIMARY KEY. Значения первичного ключа уникально идентифицируют строки в таблице. Лишь один первичный ключ может быть определен для таблицы.
Уникальный ключ	Столбец или группа столбцов, включенных в определение ограничения UNIQUE.
Внешний ключ	Столбец или группа столбцов, включенных в определение ограничения ссылочной целостности.
Адресуемый ключ	Уникальный или первичный ключ той же самой или другой таблицы, на который ссылается внешний ключ.

9. Типы данных ORACLE

Типы данных ORACLE, которые могут использоваться в определениях столбцов:

- CHAR
- VARCHAR2
- VARCHAR
- NUMBER
- DATE
- LONG
- RAW
- LONG RAW
- ROWID
- MLSLABEL

Символьные типы данных

Типы данных CHAR и VARCHAR2 хранят алфавитно-цифровые данные; в столбце одного из этих типов данных можно хранить любые символы. Символьные данные хранятся как строки символов, где байтовые значения соответствуют схеме кодирования символов (обычно называемой набором символов или кодовой страницей); набор символов базы данных устанавливается при создании базы данных и никогда не изменяется. Примером набора символов служат 7-битовый ASCII (американский стандартный код для обмена информацией). ORACLE поддерживает как однобайтовые, так и мультибайтовые схемы кодирования.

Тип данных CHAR

Тип данных CHAR хранит строки ФИКСИРОВАННОЙ длины. При создании таблицы со столбцом CHAR для этого столбца задается длина (в байтах, а не в символах) от 1 до 255 (по умолчанию 1). Затем ORACLE гарантирует соблюдение следующих правил:

- При вставке или обновлении строки в таблице значение в столбце CHAR имеет фиксированную длину.
- Если входное значение короче, оно дополняется пробелами до фиксированной длины.
- Если входное значение длиннее за счет хвостовых пробелов, то лишние пробелы отсекаются до фиксированной длины.
- Если входное значение слишком длинно, ORACLE возвращает ошибку.
  ORACLE сравнивает значения CHAR, используя ДОПОЛНЯЮЩУЮ СЕМАНТИКУ сравнения.

Тип данных VARCHAR2

Тип данных VARCHAR2 хранит символьные строки переменной длины. При создании таблицы со столбцом VARCHAR2 для этого столбца задается максимальная длина (в байтах, а не в символах) от 1 до 2000. Для каждой строки, значение столбца VARCHAR2 записывается как поле переменной длины (если входное значение превышает максимальную длину столбца, ORACLE возвращает ошибку). Например, предположим, что столбец объявлен с типом VARCHAR2 и максимальной длиной 50 символов. Если входное значение для этого столбца имеет длину 10 символов, то (в однобайтовом наборе символов) значение столбца в строке будет иметь длину 10 символов (10 байт), а не 50.

ORACLE сравнивает значения VARCHAR2, используя НЕДОПОЛНЯЮЩУЮ СЕМАНТИКУ сравнения.

Тип данных VARCHAR

Тип данных VARCHAR в настоящее время является синонимом типа данных VARCHAR2. Однако в будущей версии ORACLE тип данных VARCHAR будет хранить строки символов переменной длины с иной семантикой сравнения. Поэтому используйте тип данных VARCHAR2 для символьных строк переменной длины (ORACLE7 (tm) Server Concepts Manual).

Выбор символьного типа данных

Решая, какой тип данных выбрать для столбца символьных данных, рассмотрите следующие соображения:

- Семантика сравнения
  Используйте тип данных CHAR, если вам требуется совместимость с ANSI в семантике сравнения, т.е. корда хвостовые пробелы не должны учитываться при сравнении строк символов. Используйте VARCHAR2, если хвостовые пробелы должны учитываться при сравнениях.

- Использование памяти
  Для более эффективного хранения данных используйте тип данных VARCHAR2. Тип данных CHAR дополняет значения пробелами и сохраняет хвостовые пробелы вплоть до фиксированной длины для всех значений столбца, тогда как VARCHAR2 хранит только значащие символы значения.

- Будущая совместимость
  Типы данных CHAR и VARCHAR2 всегда будут полностью поддерживаться. В настоящее время тип данных VARCHAR автоматически соответствует типу данных VARCHAR2, однако этот тип данных зарезервирован для будущего использования.

Длины столбцов для символьных типов данных и наборов символов NLS

Средство поддержки национальных языков ORACLE (NLS) позволяет использовать для символьных типов данных разнообразные наборы символов. При специфицировании длины столбца для символьного типа данных следует принимать во внимание размеры символов. Например, некоторые символы могут занимать один байт, другие – два байта, и т.д. Это соображение должно рассматриваться при оценке памяти для таблиц, столбцы которых содержат символьные данные.

Тип данных NUMBER

Тип данных NUMBER используется для хранения чисел с фиксированной и плавающей точкой. Возможно хранить данные практически неограниченной размерности (точностью до 38 цифр), причем гарантируется переносимость между любыми операционными системами, которые поддерживает ORACLE. Тип данных NUMBER позволяет хранить следующие числа:

- ✓ положительные числа в интервале от 1 x 10**-130 до 9.99..9 x 10**125 (с точностью до 38 значащих цифр)
- ✓ отрицательные числа в интервале от -1 x 10**-130 до 9.99..9 x 10**125 (с точностью до 38 значащих цифр)
- ✓ ноль
- ✓ положительную и отрицательную бесконечность (генерируются только при импорте из базы данных версии 5)

Для числовых столбцов можно просто указать NUMBER, например: ***имя_столбца NUMBER*** или можно указать ТОЧНОСТЬ (общее число цифр) и МАСШТАБ (число цифр справа от десятичной точки): ***имя_столбца NUMBER (точность, масштаб)***. Если точность не указана, столбец хранит значения так, как они задаются. Если не указан масштаб, он считается нулевым. ORACLE гарантирует переносимость чисел с точностью, не превышающей 38 цифр. Вы можете указать масштаб и не указывать точность: ***имя_столбца NUMBER (*, масштаб)***. В этом случае поддерживаются точность 38 цифр и заданный масштаб.

При задании числовых полей рекомендуется явно указывать точность и масштаб; это обеспечивает возможность дополнительной проверки данных на входе.

Влияние показателя масштаба на хранение числовых данных

Входные данные	Тип столбца	Хранится как
7,456,123.89	NUMBER	**7456123.89**
7,456,123.89	NUMBER(*,1)	**7456123.9**
7,456,123.89	NUMBER(9)	**7456124**
7,456,123.89	NUMBER(9,2)	**7456123.89**
7,456,123.89	NUMBER(9,1)	**7456123.9**
7,456,123.8	NUMBER(6)	**ошибка: превышена точность**
7,456,123.89	**NUMBER(7,-2)**	**7456100**

Если специфицирован отрицательный масштаб, то действительные данные округляются до указанного количества цифр слева от десятичной точки. Например, спецификация (7,-2) означает округление до ближайшей сотни, как показано на табл.

Для ввода и вывода чисел стандартным умалчиваемым десятичным разделителем ORACLE является точка (например: 1234.56). (Десятичный разделитель отделяет целую часть числа от дробной.) Умалчиваемый десятичный разделитель можно изменить для инстанции с помощью параметра NLS_NUMERIC_CHARACTERS. Его можно также изменить на время сессии с помощью предложения ALTER SESSION. Для ввода чисел, которые используют десятичный разделитель, отличный от текущего умалчиваемого, можно использовать функцию TO_NUMBER.

Внутренний числовой формат

Числовые данные хранятся в формате переменной длины, начиная с байта, содержащего степень числа и знак, и до 20 байт для хранения мантиссы. (Однако лишь 38 цифр точны.) Ведущие и хвостовые нули не хранятся. Например, число 412 хранится в формате, аналогичном 4.12*10**2, с одним байтом на показатель степени (2) и тремя байтами на три значащие цифры мантиссы (4, 1, 2).

Приняв это во внимание, мы можем рассчитать размер столбца данных для конкретного числового значения данных NUMBER(p), где p – точность данного значения (масштаб не имеет значения), по следующей формуле:

1 байт *(показатель степени)* + FLOOR(p/2) + 1 байт *(мантисса)* + 1 байт *(только для отрицательных чисел, у которых число значащих цифр меньше 38)* = общее число байт данных.

Ноль, а также плюс и минус бесконечность (которые генерируются только при импорте из базы данных ORACLE версии 5) хранятся в виде уникальных представлений; ноль и минус бесконечность занимают по одному байту, плюс бесконечность – два байта.

Тип данных DATE

Тип данных DATE хранит значения в виде точек времени (т.е. дату и время). Тип данных DATE запоминает год (включая век), месяц, день, часы, минуты и секунды (после полуночи). ORACLE может хранить даты в диапазоне от 1 января 4712 года до н.э. до 31 декабря 4712 года нашей эры. Если в маске формата не указано BC (до н.э.), предполагается по умолчанию наша эра (AD).

ORACLE использует для хранения дат собственный внутренний формат. Данные дат хранятся в фиксированных полях длиной семь байт, соответствующих веку, году, месяцу, дню, часу, минуте и секунде.

Стандартный формат даты ORACLE для ввода и вывода имеет вид DD-MON-YY, например: ***'13-NOV-92'***.

Этот умалчиваемый формат даты можно изменить для инстанции с помощью параметра NLS_DATE_FORMAT. Его можно также изменить на время сессии пользователя с помощью предложения ALTER SESSION.

Для ввода дат в формате, отличном от стандартного формата дат ORACLE, используйте функцию TO_DATE с маской формата, например: ***TO_DATE ('November 13, 1992', 'Month DD, YYYY')***.

Тип данных LONG

Столбец, описанный как LONG, может содержать символьную строку переменной длины до двух гигабайт. Данные типа LONG – это текстовые данные, которые должны соответственно преобразовываться при перемещении их между различными системами.

Тип данных LONG используется в словаре данных для хранения текста определений обзоров. Столбцы, определенные как LONG, могут использоваться в списках SELECT, фразах SET предложений UPDATE и фразах VALUES предложений INSERT.

Ограничения на данные типа LONG и LONG RAW

Хотя столбцы типа LONG (и LONG RAW; см. ниже) находят много различных применений, на их использование накладываются некоторые ограничения:

- ✓ Только один столбец типа LONG допускается в таблице.
- ✓ Столбцы LONG нельзя индексировать.
- ✓ Столбцы LONG нельзя использовать в ограничениях целостности.

- ✓ Столбцы LONG нельзя использовать в фразах WHERE, GROUP BY, ORDER BY, CONNECT BY, а также с оператором DISTINCT в предложениях SELECT.
- ✓ Столбцы LONG нельзя использовать в функциях SQL (таких как SUBSTR или INSTR).
- ✓ Столбцы LONG нельзя использовать в списке SELECT подзапроса или запросов, объединяемых операторами множеств (UNION, UNION ALL, INTERSECT или MINUS).
- ✓ Столбцы LONG нельзя использовать в выражениях.
- ✓ Нельзя ссылаться на столбцы LONG при создании таблицы с помощью запроса (CREATE TABLE ... AS SELECT ...) или при вставке в таблицу (обзор) через запрос (INSERT INTO ... SELECT ...).
- ✓ Переменная или аргумент программной единицы PL/SQL не могут объявляться с типом данных LONG.

Типы данных RAW и LONG RAW

Типы данных RAW и LONG RAW используются для данных, которые не должны ни интерпретироваться ORACLE, ни преобразовываться при передаче данных между различными системами. Эти типы данных предназначены для двоичных данных, или байтовых строк. Например, LONG RAW можно использовать для хранения графики, звука, документов или массивов двоичных данных; их интерпретация зависит от их использования.

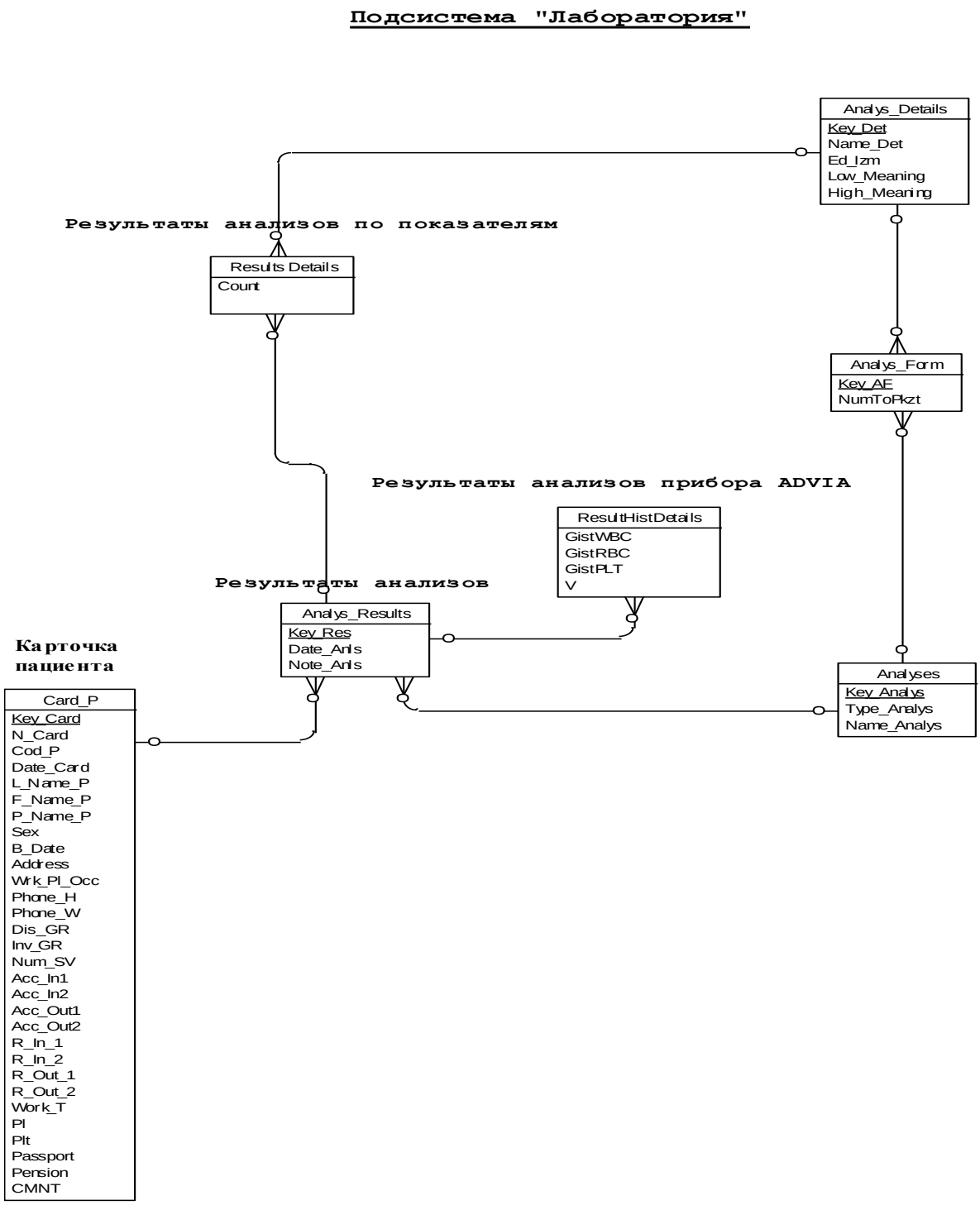
RAW эквивалентен VARCHAR2, а LONG RAW эквивалентен LONG, с тем исключением, что SQL*Net (который соединяет пользовательские сессии с инстанцией) и утилиты экспорта и импорта не выполняют преобразований при передаче данных RAW или LONG RAW. Напротив, SQL*Net и импорт/экспорт автоматически конвертируют данные CHAR, VARCHAR2 и LONG между набором символов базы данных и набором символов сессии пользователя (установленным параметром NLS_LANGUAGE или командой ALTER SESSION), если эти наборы символов различны.

Когда ORACLE автоматически преобразует данные RAW или LONG RAW в тип данных CHAR или из него, эти данные рассматриваются как шестнадцатеричные цифры, каждая из которых представляет полубайт (четыре бита). Например, один байт данных RAW с битовым представлением 11001011 вводится и отображается как 'CB'.

Данные LONG RAW не могут индексироваться, однако данные RAW можно индексировать.

11. **Создание структуры учебной базы данных с использованием СУБД Oracle.**

В качестве учебной БД для последующего создания в ORACLE, используется следующая нормализованная БД:



SQL-скрипт создания таблиц подсистемы «Лаборатория»

```
-- ============================================================
--   Table: CARD_P
-- ============================================================
create table CARD_P
(
    KEY_CARD      INTEGER               not null,
    N_CARD        INTEGER               null    ,
    COD_P         INTEGER               null    ,
    DATE_CARD     DATE                  null    ,
    L_NAME_P      RAW(50)               null    ,
    F_NAME_P      RAW(50)               null    ,
    P_NAME_P      RAW(50)               null    ,
    SEX           SMALLINT              null    ,
    B_DATE        DATE                  null    ,
    ADDRESS       RAW(200)              null    ,
    WRK_PL_OCC    RAW(200)              null    ,
    PHONE_H       RAW(15)               null    ,
    PHONE_W       RAW(15)               null    ,
    DIS_GR        SMALLINT              null    ,
    INV_GR        INTEGER               null    ,
    NUM_SV        RAW(6)                null    ,
    ACC_IN1       DATE                  null    ,
    ACC_IN2       DATE                  null    ,
    ACC_OUT1      DATE                  null    ,
    ACC_OUT2      DATE                  null    ,
    R_IN_1        RAW(200)              null    ,
    R_IN_2        RAW(200)              null    ,
    R_OUT_1       RAW(200)              null    ,
    R_OUT_2       RAW(200)              null    ,
    WORK_T        SMALLINT              null    ,
    PL            SMALLINT              null    ,
    PLT           SMALLINT              null    ,
    PASSPORT      LONG                  null    ,
    PENSION       LONG                  null    ,
    CMNT          LONG                  null    ,
    constraint PK_CARD_P primary key (KEY_CARD)
)
/
comment on table CARD_P is 'Карточки пациентов'
/
comment on column CARD_P.KEY_CARD is 'Код пациента'
/
comment on column CARD_P.N_CARD is 'номер карточки'
/
comment on column CARD_P.COD_P is 'Код пациента'
/
comment on column CARD_P.DATE_CARD is 'Дата заполнения карточки'
/
comment on column CARD_P.L_NAME_P is 'Фамилия'
/
comment on column CARD_P.F_NAME_P is 'Имя'
/
comment on column CARD_P.P_NAME_P is 'Отчество'
/
```

```
comment on column CARD_P.SEX is 'Пол'
/
comment on column CARD_P.B_DATE is 'Дата рождения'
/
comment on column CARD_P.ADDRESS is 'Адрес'
/
comment on column CARD_P.WRK_PL_OCC is 'Место работы, должность'
/
comment on column CARD_P.PHONE_H is 'Домашний телефон'
/
comment on column CARD_P.PHONE_W is 'Рабочий телефон'
/
comment on column CARD_P.DIS_GR is 'Диспансерная группа (1 - да, 2 - нет)'
/
comment on column CARD_P.INV_GR is 'Категория инвалидности'
/
comment on column CARD_P.NUM_SV is 'Номер льготного удостоверения'
/
comment on column CARD_P.ACC_IN1 is 'Acc_In1'
/
comment on column CARD_P.ACC_IN2 is 'Acc_In2'
/
comment on column CARD_P.ACC_OUT1 is 'Acc_Out1'
/
comment on column CARD_P.ACC_OUT2 is 'Acc_Out2'
/
comment on column CARD_P.R_IN_1 is 'Причина взятия на учет 1'
/
comment on column CARD_P.R_IN_2 is 'Причина взятия на учет 2'
/
comment on column CARD_P.R_OUT_1 is 'Причина снятия с учета 1'
/
comment on column CARD_P.R_OUT_2 is 'Причина снятия с учета 2'
/
comment on column CARD_P.WORK_T is 'работает на прикрепленном предприятии (0-
да, 1- нет)'
/
comment on column CARD_P.PL is 'Место жительства (город, село)'
/
comment on column CARD_P.PLT is 'Проживает в раене обслуживания или нет'
/
comment on column CARD_P.PASSPORT is 'Паспортные данные'
/
comment on column CARD_P.PENSION is '№ пенсионного удостоверения'
/
comment on column CARD_P.CMNT is 'Примечание'
/

-- ============================================================
--   Table: ANALYSES
-- ============================================================
create table ANALYSES
(
    KEY_ANALYS    INTEGER                not null,
    TYPE_ANALYS   SMALLINT               null    ,
    NAME_ANALYS   RAW(200)               null    ,
```

```
    constraint PK_ANALYSES primary key (KEY_ANALYS)
)
/
comment on table ANALYSES is 'Справочник анализов'
/
comment on column ANALYSES.KEY_ANALYS is 'Ключ записи'
/
comment on column ANALYSES.TYPE_ANALYS is 'Тип анализа (лабораторный или
инструментальный)'
/
comment on column ANALYSES.NAME_ANALYS is 'Наименование анализа'
/

-- ============================================================
--   Table: ANALYS_DETAILS
-- ============================================================
create table ANALYS_DETAILS
(
    KEY_DET       INTEGER               not null,
    NAME_DET      RAW(200)              null    ,
    ED_IZM        RAW(20)               null    ,
    LOW_MEANING   NUMBER(12,5)          null    ,
    HIGH_MEANING  NUMBER(12,5)          null    ,
    constraint PK_ANALYS_DETAILS primary key (KEY_DET)
)
/
comment on table ANALYS_DETAILS is 'Справочник показателей анализов'
/
comment on column ANALYS_DETAILS.KEY_DET is 'Ключ записи'
/
comment on column ANALYS_DETAILS.NAME_DET is 'Название показателя'
/
comment on column ANALYS_DETAILS.ED_IZM is 'Единица измерения'
/
comment on column ANALYS_DETAILS.LOW_MEANING is 'Нижний предел нормы'
/
comment on column ANALYS_DETAILS.HIGH_MEANING is 'Верхний предел нормы'
/

-- ============================================================
--   Table: ANALYS_RESULTS
-- ============================================================
create table ANALYS_RESULTS
(
    KEY_RES       INTEGER               not null,
    KEY_CARD      INTEGER               null    ,
    KEY_ANALYS    INTEGER               null    ,
    DATE_ANLS     DATE                  null    ,
    NOTE_ANLS     CHAR(30)              null    ,
    constraint PK_ANALYS_RESULTS primary key (KEY_RES)
)
/
comment on table ANALYS_RESULTS is 'Результаты анализов'
/
comment on column ANALYS_RESULTS.KEY_RES is 'Ключ записи'
/
```

```
comment on column ANALYS_RESULTS.KEY_CARD is 'Код пациента'
/
comment on column ANALYS_RESULTS.KEY_ANALYS is 'Ключ записи'
/
comment on column ANALYS_RESULTS.DATE_ANLS is 'Дата проведения анализа'
/
comment on column ANALYS_RESULTS.NOTE_ANLS is 'Примечание'
/
-- ============================================================
--   Index: RELATION_640_FK
-- ============================================================
create index RELATION_640_FK on ANALYS_RESULTS (KEY_CARD asc)
/
-- ============================================================
--   Index: RELATION_635_FK
-- ============================================================
create index RELATION_635_FK on ANALYS_RESULTS (KEY_ANALYS asc)
/

-- ============================================================
--   Table: ANALYS_FORM
-- ============================================================
create table ANALYS_FORM
(
    KEY_AF        INTEGER               not null,
    KEY_ANALYS    INTEGER               null    ,
    KEY_DET       INTEGER               null    ,
    NUM_PKZT      SMALLINT              null    ,
    constraint PK_ANALYS_FORM primary key (KEY_AF)
)
/
comment on table ANALYS_FORM is 'Готовые формы анализов'
/
comment on column ANALYS_FORM.KEY_AF is 'Ключ записи готовой формы анализов'
/
comment on column ANALYS_FORM.KEY_ANALYS is 'Ключ записи'
/
comment on column ANALYS_FORM.KEY_DET is 'Ключ записи'
/
comment on column ANALYS_FORM.NUM_PKZT is 'Номер показателя в анализе'
/
-- ============================================================
--   Index: RELATION_1860_FK
-- ============================================================
create index RELATION_1860_FK on ANALYS_FORM (KEY_ANALYS asc)
/
-- ============================================================
--   Index: RELATION 1859 FK
-- ============================================================
create index RELATION_1859_FK on ANALYS_FORM (KEY_DET asc)
/
-- ============================================================
--   Table: RES_DET
-- ============================================================
create table RES_DET
(
```

```
    KEY_RES       INTEGER               null    ,
    KEY_DET       INTEGER               null    ,
    COUNT         NUMBER(12,5)          null
)
/
comment on table RES_DET is 'Результаты анализов'
/
comment on column RES DET.KEY RES is 'Ключ записи'
/
comment on column RES_DET.KEY_DET is 'Ключ записи'
/
comment on column RES_DET.COUNT is 'Содержание данных по анализу'
/
-- ============================================================
--   Index: RELATION_3085_FK
-- ============================================================
create index RELATION_3085_FK on RES_DET (KEY_RES asc)
/
-- ============================================================
--   Index: RELATION_3082_FK
-- ============================================================
create index RELATION_3082_FK on RES_DET (KEY_DET asc)
/

-- ============================================================
--   Table: RES_HIST_DET
-- ============================================================
create table RES_HIST_DET
(
    KEY_RES       INTEGER               null    ,
    WBC           SMALLINT              null    ,
    RBC           SMALLINT              null    ,
    PLT2          SMALLINT              null    ,
    V             SMALLINT              null
)
/
comment on table RES_HIST_DET is 'Результаты гистограмм прибора ADVIA'
/
comment on column RES_HIST_DET.KEY_RES is 'Ключ записи'
/
comment on column RES_HIST_DET.WBC is 'Гистограмма WBC, количество клеток
(значение по оси Y)'
/
comment on column RES_HIST_DET.RBC is 'Гистограмма RBC, количество клеток
(значение по оси Y)'
/
comment on column RES_HIST_DET.PLT2 is 'Гистограмма PLT, количество клеток
(значение по оси Y)'
/
comment on column RES_HIST_DET.V is 'Объем клетки (номер значения по оси X)'
/
-- ============================================================
--   Index: RELATION_8898_FK
-- ============================================================
create index RELATION_8898_FK on RES_HIST_DET (KEY_RES asc)
/
```

```
alter table ANALYS_RESULTS
    add constraint FK_ANALYS_R_RELATION__CARD_P foreign key  (KEY_CARD)
       references CARD_P (KEY_CARD)
/
alter table ANALYS_RESULTS
    add constraint FK_ANALYS_R_RELATION__ANALYSES foreign key  (KEY_ANALYS)
       references ANALYSES (KEY_ANALYS)
/
alter table ANALYS_FORM
    add constraint FK_ANALYS_F_RELATION__ANALYSES foreign key  (KEY_ANALYS)
       references ANALYSES (KEY_ANALYS)
/
alter table ANALYS_FORM
    add constraint FK_ANALYS_F_RELATION__ANALYS_D foreign key  (KEY_DET)
       references ANALYS_DETAILS (KEY_DET)
/
alter table RES_DET
    add constraint FK_RES_DET_RELATION__ANALYS_R foreign key  (KEY_RES)
       references ANALYS_RESULTS (KEY_RES)
/
alter table RES_DET
    add constraint FK_RES_DET_RELATION__ANALYS_D foreign key  (KEY_DET)
       references ANALYS_DETAILS (KEY_DET)
/
alter table RES_HIST_DET
    add constraint FK_RES_HIST_RELATION__ANALYS_R foreign key  (KEY_RES)
       references ANALYS_RESULTS (KEY_RES)
/
```