ТИПЫ ДАННЫХ

Символьные типы данных

Ниже перечислены символьные типы данных в Oracle/PLSQL:

|  |  |  |
| --- | --- | --- |
| Типы данных | Размер | Описание |
| char(размер) | Максимальный размер 2000 байт. | Где размер - количество символов фиксированной длины. Если сохраняемое значение короче, то дополняется пробелами; если длиннее, то выдается ошибка. |
| nchar(размер) | Максимальный размер 2000 байт. | Где размер - количество символов фиксированной длины в кодировке Unicode. Если сохраняемое значение короче, то дополняется пробелами; если длиннее, то выдается ошибка. |
| nvarchar2(размер) | Максимальный размер 4000 байт. | Где размер – количество сохраняемых символов в кодировке Unicode переменной длины. |
| varchar2(размер) | Максимальный размер 4000 байт. Максимальный размер в PLSQL 32KB. | Где размер – количество сохраняемых символов переменной длины. |
| long | Максимальный размер 2GB. | Символьные данные переменной длины. |
| raw | Максимальный размер 2000 байт. | Содержит двоичные данные переменной длины |
| long raw | Максимальный размер 2GB. | Содержит двоичные данные переменной длины |

Применение: Oracle 9i, Oracle 10g, Oracle 11g, Oracle 12c

Числовые типы данных

Ниже приведены числовые типы данных в Oracle/PLSQL:

|  |  |  |
| --- | --- | --- |
| **Типы данных** | **Размер** | **Описание** |
| number(точность,масштаб) | Точность может быть в диапазоне от 1 до 38. Масштаб может быть в диапазоне от -84 до 127. | Например,number (14,5) представляет собой число, которое имеет 9 знаков до запятой и 5 знаков после запятой. |
| numeric(точность,масштаб) | Точность может быть в диапазоне от 1 до 38. | Например, numeric(14,5) представляет собой число, которое имеет 9 знаков до запятой и 5 знаков после запятой. |
| dec(точность,масштаб) | Точность может быть в диапазоне от 1 до 38. | Например, dec (5,2) - это число, которое имеет 3 знака перед запятой и 2 знака после . |
| decimal(точность,масштаб) | Точность может быть в диапазоне от 1 до 38. | Например, decimal (5,2) - это число, которое имеет 3 знака перед запятой и 2 знака после . |
| PLS\_INTEGER | Целые числа в диапазоне от -2,147,483,648 до 2,147,483,647 | Значение PLS\_INTEGER требуют меньше памяти и быстрее значений NUMBER |

Применение: Oracle 9i, Oracle 10g, Oracle 11g, Oracle 12c

Дата/время типы данных

Ниже приведены типы данных дата/время в Oracle/PLSQL:

|  |  |  |
| --- | --- | --- |
| **Типы данных** | **Размер** | **Описание** |
| date | date может принимать значения от 1 января 4712 года до н.э. до 31 декабря 9999 года нашей эры. |  |

Применение: Oracle 9i, Oracle 10g, Oracle 11g, Oracle 12c

Большие объекты (LOB) типы данных

Ниже перечислены типы данных LOB в Oracle/PLSQL:

|  |  |  |
| --- | --- | --- |
| **Типы данных** | **Размер** | **Описание** |
| bfile | Максимальный размер файла 4 ГБ. | Файл locators, указывает на двоичный файл в файловой системе сервера (вне базы данных). |
| blob | Хранит до 4 ГБ двоичных данных. | Хранит неструктурированные двоичные большие объекты. |
| clob | Хранит до 4 ГБ символьных данных. | Хранит однобайтовые и многобайтовые символьные данные. |
| nclob | Хранит до 4 ГБ символьных текстовых данных. | Сохраняет данные в кодировке unicode. |

Применение: Oracle 9i, Oracle 10g, Oracle 11g, Oracle 12c

Rowid тип данных

Ниже перечислены типы данных Rowid в Oracle/PLSQL:

|  |  |  |
| --- | --- | --- |
| **Типы данных** | **Формат** | **Описание** |
| rowid | Формат строки: BBBBBBB.RRRR.FFFFF, Где BBBBBBB - это блок в файле базы данных; RRRR - строка в блоке; FFFFF - это файл базы данных. | Двоичные данные фиксированной длины. Каждая запись в базе данных имеет физический адрес или идентификатор строки (rowid). |

Булевы (BOOLEAN) типы данных

|  |  |  |
| --- | --- | --- |
| **Типы данных** | **Формат** | **Описание** |
| BOOLEAN | TRUE или FALSE. Может принимать значение NULL | Хранит логические значения, которые вы можете использовать в логических операциях. |

Применение: Oracle 9i, Oracle 10g, Oracle 11g, Oracle 12c

ФУНКЦИИ И КЛЮЧЕВЫЕ СЛОВА DB ORACLE

DESCRIBE – показывает служебную информацию о колонках в таблице (название, NULL/NOTNULL, типы данных)

Структура:

DESC*RIBE schema*.table\_name;

Пример:

DESC students;

DISTINCT – возвращает уникальные значения.

Структура запроса:

SELECT *DISTINCT* column(s) FROM table

Пример:

SELECT DISTINCT job\_id FROM job\_history;

SELECT DISTINCT job\_id, department\_id FROM job\_history; (Будет выводить информацию уникальную по всем выбранным столбцам)

DUAL – позволяет что-то вывести.

Пример запроса:

SELECT 2 + 1 FROM DUAL;

CHARACTER FUNCTIONS

LIKE – оператор работы с текстовыми данными

Примеры:

SELECT \* FROM employees WHERE first\_name LIKE ‘S%’; (выводит все имена, которые начинаются на S, процент означает любое количество символов)

SELECT \* FROM employees WHERE first\_name LIKE ‘A%r’; (Aleksandr)

SELECT \* FROM employees WHERE first\_name LIKE ‘D\_n’; (' \_' – данный знак заменяет символ между буквами, в данном случае выведется только Den);

SELECT \* FROM employees WHERE first\_name LIKE ‘\_\_n%’;

SELECT \* FROM employees WHERE first\_name LIKE ‘\\_k%’ escape ‘\’; (экранирование специального символа)

ORDER BY – сортирует данные от меньшего к большему по умолчанию

Пример:

SELECT first\_name, salary FROM employees ORDER BY salary;

SELECT first\_name, salary FROM employees ORDER BY salary DESC; (Сортировка от большего к меньшему)

SELECT first\_name, salary FROM employees ORDER BY salary DESC nulls last; (c null в конце)

SELECT first\_name, salary FROM employees ORDER BY salary DESC nulls first; (c null в начале)

Пример с несколькими колонками:

SELECT job\_id, first\_name, last\_name, salary, hire\_date

FROM employees

ORDER BY job\_id desc, last\_name, 4 desc;

Сначала идёт сортировка по первому столбцу, если значения одинаковые, то идёт сравнение по данным следующего столбца.

LOWER(str) – переводит в нижний регистр входящее сообщение

Пример:

SELECT LOWER(first\_name) FROM employees;

UPPER(str) – переводит в верхний регистр входящее сообщение

Пример:

SELECT LOWER(first\_name) FROM employees;

INITCUP(str) – делает первую букву каждого слова заглавной

SELECT INITCUP(‘Test testovskiy init cup’) FROM dual;

CONCAT(str1, str2) – сложение строк (||)

SELECT CONCAT(‘Test testovskiy init cup’, ‘ Test’) FROM dual;

LENGTH(str) –узнать длину строки

SELECT LENGTH(‘Test’) FROM DUAL;

LPAD(str, n, p) –добавить определённые символы слева

str – начальная строка

n – конечная длинна строки

p – символы, которые необходимо вставить

SELECT LPAD(‘Test’, 7, ‘k’) FROM DUAL; (kkkTest)

RPAD(str, n, p) –добавить определённые символы слева

str – начальная строка

n – конечная длинна строки

p – символы, которые необходимо вставить

SELECT RPAD(‘Test’, 10, ‘$’) FROM DUAL; (Test$$$$$$)

TRIM(*{trailing | leading | both} trimstring from* s) – удаляет символ сначала или конца

Trailing – удаление с конца текста

Leading – удалит с начала текста

Both – удалит с начала и конца текста

Trimstring – текст, который надо срезать

From s – откуда срезать

Примеры:

SELECT TRIM(trailing ‘q’ from ‘Testqqqqqqqqqq’) from DUAL; (Test)

SELECT TRIM(leading ‘q’ from ‘qqqTestqqqqqqqqqq’) from DUAL; (Testqqqqqqqqqq’)

SELECT TRIM(*both* ‘q’ from ‘qqqTestqqqqqqqqqq’) from DUAL; (Test)

INSTR(s, search string, *start position, Nth occurrence*) – возвращает индекс символа или слова в строке

s – строка

search string – искомый текст (что будем искать в строке)

start position – позиция для начала поиска;

Nth occurrence – N-ое появление

Примеры:

SELECT \* FROM employees WHERE INSTR(job\_id, ‘PROG’) = 4;

SELECT INSTR(‘Test’, ‘e’) FROM DUAL; (2)

SELECT INSTR(LOWER(‘TestTestovskiy’), ‘t’, 2, 2) FROM DUAL; (5)

SUBSTR (s, start position*, number of characters*) – возвращает подстроку из строки

Примеры:

SELECT email, SUBSTR(email, 4) FROM employees;

SELECT SUBSTR(‘TestTov’, 4) FROM employees; (test)

SELECT SUBSTR(‘TestTov’, 4, 3) FROM employees; (Tov)

REPLASE(s, search item*, replacement item*) – заменяет строку/букву в строке

Если не указывается третий параметр, то работает как удаление буквы из слова.

Пример:

SELECT REPLASE(‘Chto horoshego?’, ‘o’, ‘tt’) FROM DUAL; (Chttt httrttshegtt)

SELECT REPLASE(‘Chto horoshego?’, ‘o’) FROM DUAL; (Cht hrsheg)

NUMERIC FUNCTIONS

ROUND (n, *precition*) – округление числа (число, точность)

Пример:

SELECT ROUND(3.14) FROM DUAL; (3)

SELECT ROUND(3.14, 1) FROM DUAL; (3.1);

SELECT ROUND(3.17, 1) FROM DUAL; (3.2);

SELECT ROUND(3.58745, 2) FROM DUAL; (3.59);

TRUNC (n, precition) – обрезание числа (число, точность)

Пример:

SELECT ROUND(3.14) FROM DUAL; (3)

SELECT ROUND(3.14, 1) FROM DUAL; (3.1);

SELECT ROUND(3.17, 1) FROM DUAL; (3.1);

SELECT ROUND(3.58745, 2) FROM DUAL; (3.58);

MOD (dividend, divisor) – остаток от деления (делимое, делитель)

Примеры:

SELECT MOD(7, 3) FROM DUAL; (1)

SELECT MOD(8, 3) FROM DUAL; (2)

SELECT MOD(9, 3) FROM DUAL; (0)

DATE FUNCTIONS

SYSDATE – возвращает время db сервера.

Пример:

SELECT SYSDATE FROM DUAL;

MONTHS\_BETWEEN(start\_date, end\_date) – возвращает количество месяцев между дат

Пример:

SELECT employee\_id, MONTHS\_BETWEEN(end\_date, start\_date) FROM job\_history;

ADD\_MONTHS(date, number\_of\_months) – прибавляет некоторое месяцы к дате

Пример:

SELECT end\_date, ADD\_MONTHS(end\_date, 4) FROM job\_history;

NEXT\_DAY(date, day\_of\_the\_week) – возвращает следующую дату дня недели

date – дата;

day\_of\_the\_week – день недели дату которого необходимо получить

Пример:

SELECT NEXT\_DAY (sysdate, 3) FROM DUAL;

LAST\_DAY(date) – возвращает последний день месяца

Пример:

SELECT LAST\_DAY(SUSDATE) FROM DUAL;

ROUND(date, date precision formant) – округление даты

date – дата

date precision formant – точность округления (

Век – СС

Год – YYYY

Четверть – Q

Месяц – MM

Неделя – W

День – DD

Час – HH

Минута – MI

)

Пример:

SELECT hire\_date, ROUND(hire\_date, ‘CC’) FROM employees WHERE employee\_id IN (120, 121);

TRUNC (date) – округление с отбрасываниемы

date – дата

date precision formant – точность округления (

Век – СС

Год – YYYY

Четверть – Q

Месяц – MM

Неделя – W

День – DD

Час – HH

Минута – MI

)

Пример:

SELECT hire\_date, TRUNC(hire\_date, ‘YYYY’) FROM employees

WHERE employee\_id IN (120, 121);

CONVERSION FUNCTIONS – конвертирование из одного типа данных в другой

TO\_CHAR(num\_to\_char, *format mask, nls\_parameters*) – переводит число или дату в текст

*format –*если понадобится, то лучше загуглить

Примеры:

SELECT TO\_CHAR(12) FROM DUAL;

(У этой функции действительно широкий функционал формата конвертации, лучше загуглить под конкретную задачу)

TO\_DATE – конвертирует текст в дату

Сигнатура:

TO\_DATE(text, *format mask, nls\_parameters*)

Примеры:

SELECT TO\_DATE(’08-MAR-19’) FROM DUAL;

SELECT TO\_DATE(’08!MAR&19’) FROM DUAL;

TO\_NUMBER (NUMBER TO DATE) – конвертация текста в число

TO\_NUMBER(text, *format mask, nls\_parameters*)

Пример:

SELECT TO\_NUMBER(‘4555’) FROM DUAL;

SELECT TO\_NUMBER(‘4555’, ‘$9999999999.99’) FROM DUAL;

GENERAL FUNCTIONS – функции, которые упрощают работу с NULL

NVL(value, ifnull) – проверяет первый параметр (value) и если он null возвращает второй иначе возвращает первый. Типы данных в скобках должны быть одинаковы.

SELECT NV(18, 19) FROM DUAL; (18)

NVL2(value, ifnotnull, if null) – если value не равно null, то возвращает ifnotnull иначе возвращает ifnull;

Пример:

SELECT NVL2(17, 18, 19) FROM DUAL; (18)

SELECT NVL2(null, 18, 19) FROM DUAL; (19)

NULLIF (value1, value2) – возвращает null если value1 = value2 и возвращает value1 если value1 != value2.

Пример:

SELECT NULLIF(17, 17) FROM DUAL; (null)

SELECT NULLIF(17, 18) FROM DUAL; (17);

COALESCE(value1, value2*..valueN*) – возвращает первое не null значение в вложенных в функцию values.

Пример:

SELECT COALESCE(null, null, ‘ok’, ‘hi’) FROM DUAL; (ok)

SELECT COALESCE(null, null, null, null) FROM DUAL; (null)

COMDITIONAL FUNCTIONS – функции с условием

DECODE(expr, comp1, iftrue1, comp2, iftrue2…compN, iftrueN,iffalse) – сравнивает первый параметр с другими. Если первый параметр равен второму, то возвращается третий.

Пример:

SELECT DECODE(3\*4, 12, ‘dvenadcat’) FROM DUAL; (dvenadcat)

SELECT DECODE(3\*4, 13, ‘dvenadcat’) FROM DUAL; (null)

SELECT DECODE(3\*5, 13, ‘dvenadcat’, 14, ‘chetyrnadcat’, 15, ‘puatnadcat’) FROM DUAL; (puatnadcat)

SELECT DECODE(3\*4, 13, ‘dvenadcat’, ‘nedvenadcat’) FROM DUAL; (nedvenadcat)

CASE simple – тоже самое, что и DECODE, только другой синтаксис

CASE expt

WHEN compl1 THEN iftrue1

*WHEN compl2 THEN iftrue2*

*ELSE iffalse*

END

Пример:

SELECT

CASE 3\*4

WHEN 12 THEN 100

END FROM DUAL; (100)

SELECT

CASE 3\*4

WHEN 13 THEN 100

ELSE 12

END FROM DUAL; (12)

SELECT

CASE 3\*4

WHEN 13 THEN 100

END FROM DUAL; (null)

CASE searched – условие пишется после слова WHEN

CASE

WHEN compl1 THEN iftrue1

*WHEN compl2 THEN iftrue2*

*ELSE iffalse*

END

Примеры:

SELECT

CASE

WHEN 3\*4=12 THEN 100

END

FROM DUAL; (100)

SELECT

CASE

WHEN 3\*4=13 THEN 100

ELSE 12

END

FROM DUAL; (12)

GROUP FUNCTIONS

COUNT – возвращает количество

COUNT(*{ \* | {DICTINCT | ALL}* expression})

Пример:

SELECT COUNT(\*) FROM employees; (возвращает количество строк)

SELECT COUNT(\*) FROM employees WHERE salary < 5000;

SELECT COUNT(salary) FROM employees; (возвращает количество строк)

SELECT COUNT(DISTINCT commission\_pct) FROM employees; (возвращает количество уникальных строк)

SUM ({DISTINCT|ALL} expretion) – суммирует значения в таблице по какой-то колонке

Пример:

SELECT SUM(salary) FROM employees;

SELECT SUM(DISTINCT salary) FROM employees; (только уникальные зп)

AVG({DISTINCT |ALL} expression) – возвращает среднее арифметическое;

Пример:

SELECT AVG(salary) FROM employees;

SELECT AVG(salary) FROM employees WHERE job\_id = ‘IT\_PROG’;

SELECT AVG(DISTINCT salary) FROM employees;

MIN/MAX ({DISTINCT|ALL}expression) – MIN возвращает минимальное значение, MAX возвращает максимальное значение

Примеры:

SELECT MIN(salary) MAX(salary) FROM employees WHERE department\_id = 50;

GROUP BY – группирует данные в таблице

Примеры:

SELECT department\_id , COUNT(\*) FROM employees GROUP BY department\_id ORDER BY 1, 2;

(Вернёт сгруппированную и отсортированную информацию по department\_id)

SELECT department\_id , COUNT(\*), MIN(salary) FROM employees GROUP BY department\_id ORDER BY 1, 2;

SELECT job\_id, MIN(salary), MAX(salary)

FROM employees

GROUP BY job\_id;

JOIN – объединение таблиц

INNER JOIN (NATURAL JOIN) – выводит только ту информацию, которая есть в объединяемых таблицах (region\_id) поиск такого столба выполняется автоматически (столб с одинаковым именем)

Структура:

SELECT column(s)

FROM table\_1

NATURAL JOIN

table\_2

Примеры:

SELECT \* FROM regions NATURAL JOIN countries; (region\_id)

SELECT first\_name, last\_name, salary, department\_name, department\_id FROM employees

NATURAL JOIN departments; (department\_id, manadger\_id)

SELECT c.contry\_name, c.contry\_id, r.region\_name, region\_id

FROM regions r NATURAL JOIN contries c

WHERE region\_name = ‘Europe’;

NATURAL JOIN WITH USIING – с указанием столба по которому необходимо объединить информацию в таблице

Структура запроса:

SELECT column(s) FROM table\_1 JOIN table\_2 USING(column(s));

Примеры:

SELECT first\_name, last\_name, salary, department\_name, department\_id FROM employees e

JOIN departments d USING (department\_id);

SELECT \* FROM regions JOIN countries USING (region\_id);

SELECT first\_name, last\_name, jh.job\_id, start\_date, end\_date FROM employees

JOIN job\_history jh USING (employee\_id, department\_id);

JOIN WITH ON – join с выбором колонок из одной и другой таблиц для объединения

Структура запроса:

SELECT column(s) FROM table\_1 JOIN table\_2 ON (column1 = column2);

Примеры:

SELECT first\_name, last\_name, jh.job\_id, start\_date, end\_date FROM employees e

JOIN job\_history jh ON (e.employee\_id = jh. employee\_id);

SELECT \* FROM departments JOIN regions ON (region\_id = department\_id);

SELECT first\_name, last\_name, jh.job\_id, start\_date, end\_date FROM employees e

JOIN job\_history jh

ON (e.employee\_id = jh.employee\_id AND e.department\_id = jh.department\_id);

Объединение нескольких таблиц

SELECT \* FROM locations NATURALS JOIN countries NATURAL JOIN regions; (объединение произойдёт на основе regions\_id)

SELECT \* FROM locations JOIN countries USING (country\_id) JOIN regions USING (region\_id);

SELECT first\_name, last\_name, jh.job\_id, start\_date, end\_date, department\_name

FROM employees e JOIN job\_history jh ON (e.employee\_id = jh.employee\_id)

JOIN departments d ON (jh.department\_id = d.department\_id);

NONEQUIJOIN with ON – объединение таблиц по неравным данным

SELECT first\_name, salary, min\_salary, max\_salary

FROM employees e JOIN jobs j ON (e,job\_id = j.job\_id AND salary\*2 < max\_salary);

SELECT first\_name, salary, min\_salary, max\_salary

FROM employees e JOIN jobs j ON (e,job\_id = j.job\_id AND salary = max\_salary);

SELECT first\_name, salary, min\_salary, max\_salary

FROM employees e JOIN jobs j ON (e,job\_id = j.job\_id AND salary < max\_salary);

SELECT first\_name, salary, min\_salary, max\_salary

FROM employees e JOIN jobs j ON (e,job\_id = j.job\_id AND salary between min\_salary + 2000

AND max\_salary - 3000);

OUTER JOIN – выводит даже те строки, которые не попадают под критерии выбора колонки (под объединение) Делятся на LEFT, RIGHT и FULL

LEFT OUTER JOIN – выводится информация по INNER JOIN и прибавляются все строки из левой таблицы для которых условие объединение по столбцу не было выполнено

Структура:

SELECT column(s) FROM table\_1 LEFT OUTER JOIN table\_2 ON (column1 = column2);

Пример:

SELECT first\_name, last\_name, salary, department\_name

FROM employees e LEFT JOIN departments d ON (e.department\_id = d.departments\_id);

RIGTH OUTER JOIN – выводится информация по INNER JOIN и прибавляются все строки из правой таблицы для которых условие объединение по столбцу не было выполнено

Структура:

SELECT column(s) FROM table\_1 RIGTH OUTER JOIN table\_2 ON (column1 = column2);

Пример:

SELECT first\_name, last\_name, salary, department\_name

FROM employees e RIGTH JOIN departments d ON (e.department\_id = d.departments\_id);

FULL OUTER JOIN – сначала выводятся строки, которые удовлетворяют условиям объединения по колонке, а затем те, что не удовлетворяют сначала из левой таблицы, а потом из правой таблицы

Структура:

SELECT column(s) FROM table\_1 FULL OUTER JOIN table\_2 ON (column1 = column2);

Пример:

SELECT first\_name, last\_name, salary, department\_name

FROM employees e FULL JOIN departments d ON (e.department\_id = d.departments\_id);

SUBQUERY – подзапросы, которые вложены в запрос

Примеры:

SELECT first\_name, last\_name, salary FROM employees

WHERE salary > (SELECT AVG(salary) FROM employees); (выводит сотрудников у которых зп выше средней)

SELECT (SELECT MIN(min\_salary) FROM jobs) min\_zp,

(SELECT MAX(LENGTH(first\_name)) FROM employees) samoe\_dlinnoe\_imya

FROM DUAL; (выводит минимальное значение зп и самое длинное имя)

SUBQUERY SINGLE ROW – подзапрос с возвратом какого-то одного значения

Пример:

SELECT first\_name, last-name, salary FRPM employees

WHERE salary < (SELECT MAX(salary) / 5 FROM employees); (Выводит список работников чья зп меньше максимальная зп / 5);

SELECT first\_name, last\_name, salary FROM employees WHERE salary > (SELECT AVG(salary) FROM employees); (выводит работников у которых зп выше средней)

SUBQUERY MULTIPLE ROW – подзапрос, который возвращает новое представление таблицы (несколько строк) используется оператор IN

Примеры:

SELECT first\_name, last\_name, salary FROM employees

WHERE job\_id IN (SELECT job\_id FROM jobs WHERE min\_salary > 8000);

SELECT first\_name, last\_name, salary FROM employees

WHERE job\_id NOT IN (SELECT job\_id FROM jobs WHERE min\_salary > 8000);

SELECT first\_name, last\_name, salary FROM employees

WHERE salary > ANY(любой)(SELECT salary F ROM employees WHERE department\_id = 100);

CORRELATED SUBQUEY – когда подзапрос связан с внешним запросом (подзапросу необходимо постоянно получать данные из внешнего запроса)

SELECT e1.first\_name, e1.last\_name, e1.salary FROM employees e1

WHERE salary > (SELECT AVG(e2.salary) FROM employees e2

WHERE e.2department\_id = e1.department\_id);

SET операторы

Общие правила:

Количество столбцов в первом select'e и во втором должно быть одинаково

SELECT first\_name FROM employees

UNION

SELECT dep\_name FROM departments

Типы данных в объединяемых столбах должны быть одинаковы (можно по одному семейству INTEGER и NUMBER, VARCHAR2 и CHAR)

UNION ALL – берет данные из двух множеств и просто объединяет их в одно (всё в кучу)

Примеры:

SELECT \* FROM jobs WHERE job\_id like ‘%MAN%’

UNION ALL

SELECT \* FROM jobs WHERE job\_id like ‘%MAN%’; (Выдаст просто продублированные данные)

SELECT job\_id, max\_salary FROM jobs WHERE job\_id like ‘%MAN%’

UNION ALL

SELECT job\_title, min\_salary FROM jobs WHERE job\_id like ‘%MAN%’;

SELECT job\_id, max\_salary FROM jobs WHERE job\_id like ‘%MAN%’

UNION ALL

SELECT job\_title, min\_salary FROM jobs WHERE job\_id like ‘%MAN%’

ORDER BY min\_salary DESC;

UNION – объединяет два множество в одно, удаляет дубликаты и сортирует множество от больше к меньшему

SELECT \* FROM jobs WHERE job\_id like ‘%MAN%’

UNION

SELECT \* FROM jobs WHERE job\_id like ‘%MAN%’; (Выдаст просто данные как и при обычном селекте)

INTERSECT – выводит общие значения из множеств, исключает дубликаты и выводит их в отсортированном виде

Пример:

SELECT \* FROM jobs WHERE min\_salary BETWEEN 4500 and 8000

INTERSEC

SELECT \* FROM jobs WHERE max\_salary BETWEEN 10000 and 15000; (выведет только те строки, которые есть в первом запросе и во втором без дубликатов и отсортирует их)

MINUS (EXCEPT) – возвращает только те строки, которые есть в первом множестве, но нет во втором, удаляя при этом дубликаты и сортируя результат

Пример:

SELECT \* FROM jobs WHERE job\_id like ‘%MAN%’

MINUS

SELECT \* FROM jobs WHERE job\_id like ‘%MAN%’; (Выдаст пустое значение т.к. от первого отнимаем второе)

SELECT job\_id, max\_salary FROM jobs WHERE job\_id like ‘%MAN%’

MINUS

SELECT job\_title, min\_salary FROM jobs WHERE job\_id like ‘%MAN%’

ORDER BY min\_salary DESC;

DML (Data Manipulation Language)

DML позволяет осуществлять манипуляции с данными таблиц, т.е. с ее строками. Он позволяет делать выборку данных из таблиц, добавлять новые данные в таблицы, а так же обновлять и удалять существующие данные.

SELECT выбор строк из таблицы

Структура запроса:

SELECT (colum(s)\_name)

FROM table\_name;

INSERT вставка строки в таблицу

Структура запроса:

INSERT INTO table\_name

*colum(s)*

VALUES (value(s));

Примеры запроса:

INSERT INTO countries *(country\_id, country\_name, region\_id)*

VALUES (‘NR’, ‘Norway’, 1);

Можно комбинировать с функциями:

INSERT INTO countries *(country\_id, country\_name, region\_id)*

VALUES (UPPER(‘nw’), INITCAP(‘Norway’), 1);

Структура запроса с использованием SUBQUERY (вставка нескольких строк сразу):

INSERT INTO table\_name

*colum’s*

SUBQUERY;

Пример запроса с использованием SUBQUERY:

INSERT INTO new\_countries *(country\_id, country\_name, region\_id)*

SELECT country\_id, country\_name, region\_id FROM countries

WHERE country\_id < 3;

UPDATE изменение данных в таблице

Структура запроса:

UDATE table\_name

SET column(s) = value(s)

*WHERE condition(s);*

Примеры запроса:

UPDATE employees

SET salary = 10000

WHERE employees\_id = 100;

UPDATE employees

SET salary = 15000, job\_id = ‘IT\_PROG’

WHERE employees\_id = 100;

Для нескольких строк:

UPDATE employees

SET salary = salary \* 1.1

WHERE job\_id = ‘IT\_PROG’;

Для всей таблицы:

UPDATE employees

SET salary = salary \* 0.5;

Обновление информации с использованием SUBQUERY

Структура:

UDATE table\_name

SET column(s) = value(s)

*WHERE column = subquery;*

Примеры:

UPDATE employees

SET salary = 5000

WHERE department\_id

IN

(SELECT department\_id

FROM departments

WHERE department\_name = ‘Marketing’);

UPDATE employees

SET salary = (SELECT MAX(salary)) FROM employees),

hire\_date = (SELECT MIN(start\_date) FROM job\_history)

WHERE employee\_id = 180;

DELETE удаление строки из таблицы

Структура запроса (без указания WHERE удаляет все строки):

DELETE

FROM table\_name

WHERE condition(s);

Пример:

DELETE FROM employees

WHERE employee\_id = 15

OR first\_name is null;

Структура запроса c SUBQUERY

DELETE

FROM table\_name

WHERE column = subquery;

Пример запроса c SUBQUERY

DELETE FROM new\_employees

WHERE job IN

(SELECT job\_id FROM employees WHERE department\_id IN

(SELECT department\_id FROM departments WHERE manager\_id = 100));

ТРАНЗАКЦИИ (TCL transaction control language)

Транзакция – выполнение одной или последовательности команд. Транзакция открывается при изменении таблицы и заканчивается, когда выполняются команды COMMIT (в случае успеха) или ROLLBACK (в случае если какая-то из команд не прошла)

Принципы ACID

A – ATOMICITY (атомарность) либо срабатывают все изменения (выполняются все команды) либо не выполняется ни одной (в случае ошибки на какой-то команде происходит откат).

С – CONSISTENCY (согласованность) выборка информации происходит с данными таблицы, которые были заложены на момент запроса, все изменения, которые совершаются в таблице на момент выполнения запроса не учитываются

I – ISOLATION (изолированность) гарантирует, что запросы будут выполняться только с подтверждённой информацией в таблице т.е. после команды COMMIT. Изменения произведённые до команды COMMIT не будут отображаться в запросах других пользователей.

D – DURABLE (долговечность) гарантирует, что все после COMMIT’a все изменения будут видны другим пользователям и останутся в таблице, если в последующем конечно не будет выполнена команда DELETE.

Транзакция может завершиться в процессе выполнения в случаях:

Если изменилась структура таблицы или данные таблицы (Выполняется автоматический COMMIT)

Если изменились права доступа (Выполняется автоматический COMMIT)

Если потеряна связь с сервером (Выполняется автоматический ROLLBACK)

COMMIT – фиксирует все проводимые изменения в таблице

ROLLBACK – отменяет все проводимые изменения в таблице. Можно так же использовать ROLLBACK до точки сохранения SAVEPOINT, синтаксис будет следующим:

Структура: ROLLBACK TO SAVEPOINT savepoint\_name;

SAVEPOINT – точка сохранения внутри транзакции к которой можно откатиться

INSERT INTO employees VALUES (102, ‘Karl’, ‘SYSDATE’, ‘IT\_PROG’);

SAVEPOIN first;

UPDATE employees SET employee\_id = 200 WHERE employee\_id = 100;

SAVEPOIN second;

DELETE FROM employees WHERE first\_name = ‘Nina’;

ROLLBACK TO SAVEPOINT first;

DLL (Data Definition Language)

Язык DDL служит для создания и модификации структуры БД, т.е. для создания/изменения/удаления таблиц и связей.

USER

User это лицо, которое может подключиться к базе данных имеет USERNAME и PASSWORD

SCHEMA

Schema это все объекты в базе данных, которые принадлежат одному пользователю.

(контейнер для объектов, которые принадлежат user’у)

Создание таблиц

Структура команды:

CREATE TABLE *schema*.table *ORGANIZATION HEAP*

(column\_name datatype *DEFAULT expr*,

*column\_name datatype DEFAULT expr*,

…);

Пример команды:

CREATE TABLE students (

student\_id integer,

name varchar2(15),

start\_date date DEFAULT ROUND(SYSDATE),

scholarship number(6, 2),

avg\_score number(4,2) DEFAULT 5

) *ORGANIZATION HEAP*;

Создание с использованием SUBQUERY:

CREATE TABLE schema.table

AS subquery;

Пример создания с помощью SUBQUERY:

CREATE TABLE new\_empl AS

SELECT employee\_id, first\_name, last\_name, salary, department\_id

FROM employees;

CREATE TABLE new\_dep AS

SELECT department\_name, MAX(salary) max\_salary, MIN(salary) min\_salary

FROM employees e JOIN departments d ON (e.department\_id = d.department\_id)

GROUP BY department\_name;

Изменение таблиц (ALTERING TABLE)

Примеры структур запроса:

Добавить столбец:

ALTER TABLE table\_name

ADD (column\_name data\_type DEFAULT expr);

Изменить столбец:

ALTER TABLE table\_name

MODIFY (column\_name data\_type DEFAULT expr);

Удалить колонку:

ALTER TABLE table\_name

DROP COLUMN column\_name;

Сделать колонку неиспользуемой (альтернатива DROP скрывает колонку т.к. удаление занимает больше времени):

ALTER TABLE table\_name

SET UNUSED COLUMN column\_name;

Удалить неиспользованную колонку:

ALTER TABLE table\_name DROP UNUSED COLUMNS;

Изменить имя колонки:

ALTER TABLE table\_name

RENAME COLUMN column\_name1 TO column\_name2;

Сделать таблицу только для чтения

ALTER TABLE table\_name READ ONLY;

Примеры:

ALTER TABLE students ADD (course number DEFAULT 3);

ALTER TABLE students MODIFY (avg\_score number(5,3));

ALTER TABLE students DROP COLUMN scholarship;

ALTER TABLE students SET UNUSED COLUMN start\_date;

ALTER TABLE students DROP UNUSED COLUMNS;

ALTER TABLE students RENAME COLUMN student\_id TO id;

ALTER TABLE students READ ONLY;

TRUNCATE (очистка данных таблицы)

Структура команды:

TRUNCATE TABLE *schema*.table\_name

Пример:

TRUNCATE TABLE students;

Удаление таблицы

Структура команды:

DROP TABLE

*schema*.table\_name;

Пример:

DROP TABLE students;

CONSTRAINTS

UNIQUE – принуждает столбец содержать только уникальные значения. Исключение null.

Пример:

CREATE TABLE students (

id number *CONSTRAINT st\_id\_unique* UNIQUE,

name varchar2(15),

course number,

faculty\_id integer,

avg\_score number(5,2),

start\_date date,

scholarship integer

);

Пример с композитным CONSTRAINT’ом, уникальной должна быть пара:

(id = 1, name Misha)

(id = 1, name Vasya) Можно

CREATE TABLE students (

id number,

name varchar2(15),

course number,

faculty\_id integer,

avg\_score number(5,2),

start\_date date,

scholarship integer

*CONSTRAINT st\_id\_unique* UNIQUE (id, name)

);

Добавление CONSTAINT после создания таблицы:

ALTER TABLE students ADD *CONSTRAINT st\_id\_unique* UNIQUE (id);

ALTER TABLE students MODIFY (id *CONSTRAINT st\_id\_unic* UNIQUE);

Удаление CONSTRAINT’а

ALTER TABLE students DROP constraint *st\_id\_unicl;*

NOT NULL – значение в колонке не может быть null

Пример:

CREATE TABLE students (

id number,

name varchar2(15) *CONSTRAINT st\_name\_notnull*  NOT NULL,

course number,

faculty\_id integer,

avg\_score number(5,2),

start\_date date,

scholarship integer

);

Добавление CONSTAINT после создания таблицы:

ALTER TABLE students MODIFY (name *CONSTRAINT st\_name\_notnull*  NOT NULL);

PRIMARY KEY – сочетает в себе UNIQUE и NOT NULL т.е. нельзя вставлять значения null и уже существующие значения в таблице.

Пример:

CREATE TABLE students (

id number *CONSTRAINT st\_name\_primary\_key*  PRIMARY KEY,

name varchar2(15),

course number,

faculty\_id integer,

avg\_score number(5,2),

start\_date date,

scholarship integer

);

Добавление CONSTAINT после создания таблицы:

ALTER TABLE students MODIFY (id *CONSTRAINT st\_id\_primary\_key* PRIMARY KEY);

FOREIGN KEY – принуждает использовать только значения из определённого столбца таблицы-родителя или значение “null”

Пример:

CREATE TABLE faculties (

id number PRIMARY KEY,

name varchar2(15)

);

CREATE TABLE students (

id number,

name varchar2(15),

course number,

faculty\_id integer *CONSTRAINT st\_faculty\_fk* REFERENCES faculties(id),

avg\_score number(5,2),

start\_date date,

scholarship integer

);

Как выглядит INSERT:

INSERT INTO faculties

VALUES (2, Vasya, PS, должен быть существующий в таблице faculties, …);

Добавление CONSTAINT после создания таблицы:

ALTER TABLE students MODIFY (*CONSTRAINT st\_fk\_foreign* faculty\_id references faculties(id));

ALTER TABLE students MODIFY (faculty\_id *CONSTRAINT st\_fk\_foreign*  references faculties(id));

ALTER TABLE students ADD *CONSTRAINT fk* FOREIGN KEY (faculty\_id)

REFERENCES faculties(id);

ON DELETE CASECADE – применяется к полю которое ссылается на ключ другой таблицы и при удалении этого ключа удаляются все строки, которые содержат этот ключ уже ссылающейся таблицы.

Пример:

CREATE TABLE faculties (

id number PRIMARY KEY,

name varchar2(15)

);

CREATE TABLE students (

id number,

name varchar2(15),

course number,

faculty\_id integer REFERENCES faculties(id) ON DELETE CASCADE,

avg\_score number(5,2),

start\_date date,

scholarship integer

);

Если удалить факультет из таблицы, то удалятся и строки из таблицы students, которые содержат этот факультет.

ON DELETE SET NULL – обнуляет (делает null) поле таблицы students , которое указывало на удалённый факультет.

CHECK – констрейнт условия при вставке значения в поле таблицы.

Пример:

CREATE TABLE students (

id number,

name varchar2(15),

course number *CONSTRAINT ch* CHECK (course > 0 AND course < 6),

faculty\_id integer,

avg\_score number(5,2),

start\_date date,

scholarship integer

);

Модификация после создания таблицы:

ALTER TABLE students MODIFY (id CHECK (id >= 1));

Создание правила на наличие какого-либо символа в запросе:

email varchar(15) CHECK (INSTR(email, ‘@’) > 0),

ИНДЕКСЫ – ускоряют поиск по столбцу, но подходит не дял всех столбцов в основе лежит (B-tree либо bitmap)

Создание индекса:

CREATE {UNIQUE OR BITMAP} INDEX schema.index\_name ON

Schema.table\_name (column1, column2, …);

Создание индекса с возможностью добавления дубликатов:

CREATE INDEX in1 ON students(id);

Создание индекса с запретом добавления дубликатов:

CREATE UNIQUE INDEX in2 ON students(name);

Композитный индекс:

CREATE INDEX in2 ON students(id, name);

Удаление индекса:

DROP INDEX schema.index\_name;

Создание BITMAP индекса (Только в расширенной версии Oracle DB)

CREATE BITMAP INDEX b\_ind1 ON students(faculty\_id);

VIEW – это возврат команды select. Может быть простым simple или complex.

Структура запроса:

CREATE *OR REPLACE {FORCE | NOFORCE}* VIEW

*schema*.view\_name *(alias1, alias2, …)*

AS subquery

*WITH CHECK OPTION {CONSTRAINT constraint\_tname}*

*WITH READ ONLY {CONSTRAINT constraint\_name};*

Причины работы с VIEW:

Безопасность, можно дать доступ к некоторой информации из таблицы (создать view).

Создать более упрощённый вид таблицы для более опытных User’s.

Сделать оптимизированный SELECT.

Пример создания:

CREATE VIEW fin\_empl AS

SELECT employee\_id, job\_id, salary FROM employees;

CREATE VIEW fin\_empl2 AS

SELECT department\_name, SUM(salary) sum\_salary FROM employees e JOIN departmets d

ON e.department\_id = d. department\_id GROUB BY department\_name;

Аргумент FORCE создаёт вид даже если нет колонки или таблицы

Пример:

CREATE FORCE VIEW fin\_empl AS

SELECT employee\_id, job\_id, salary FROM employees\_no\_existe;

Аргумент WITH CHECK OPTIONT – создаёт представление с условием для добавление строки в него

Пример:

CREATE VIEW v202 AS SELECT \* FROM students WITH READ ONLY CONSTRAINT ogranichenir\_1;

Аргумент WITH CHECK OPTIONT дополнительное правило с запретом редактирования данных в таблице согласно условию (where course > 2).

CREATE VIEW v202 AS SELECT \* FROM students where course > 2 WITH CHECK OPTIONT;

Т.е. через представление нельзя будет добавить в таблицу значение если курс меньше двух.

Аргумент WITH READ ONLY дополнительное правило с запретом редактирования данных в таблице.

CREATE VIEW v202 AS SELECT \* FROM students WITH READ ONLY;

Изменение View (необходимо удалить перед этим старое представление):

ALTER VIEW *schema*.view\_name COMPILE;

Удаление View:

DROP VIEW *schema*.view\_name;

Типы VIEW

SIMPLE – можем делать DML команды (редактировать)

Одна таблица, не можем использовать функции, не используем агрегацию.

Пример (нет противоречий для редактирования таблицы):

CREATE VIEW fin\_empl AS

SELECT employee\_id, job\_id, salary FROM employees;

COMPLEX – можем объединять таблицы, использовать функции, использовать агрегации.

Пример (есть противоречия для редактирования таблицы):

CREATE VIEW fin\_empl2 AS

SELECT department\_name, SUM(salary) sum\_salary FROM employees e JOIN departmets d

ON e.department\_id = d. department\_id GROUB BY department\_name;

SYNONYM – альтернативное имя объекта (обычно талицы или представление)

Структура:

CREATE *PUBLIC* SYNONYM synonym\_name FOR object\_name;

Пример:

CREATE SYNONYM syn1 FOR students;

Удобно использовать если необходимо обращаться к другой базе

SELECT \* FROM hr.employees@orclpdb2; (@orclpdb2 дб линк)

CREATE PUBLIC SYNONYM syn1 FOR hr.employees@orclpdb2;

Только необходимо иметь права доступа создания.

Удалить синоним:

DROP *PUBLIC* SYNONYM synonym\_name;

SEQUENCE – генерирует уникальные числа

Структура:

CREATE SEQUENCE *schema.*sequence\_name

*INCREMENT BY number*

*START WITH number*

*{MAXVALUE number | NOMAXVALUE}*

*{MINVALUE number | NOMINVALUE}*

*{CYCLE | NOCYCLE}*

*{CACHE number | NOCACHE};*

INCREMENT BY – увеличение числа на.

Пример: если значение равно 3-м, то 1,4,7,10

START WITH – начальное значение

{MAXVALUE number | NOMAXVALUE} – указывает или не указывает верхнюю границу последовательности если использовать с CYCLE, то при достижении верхней границы начинает отсчёт с START WITH, если не указывать CYCLE, то при достижении верхней границы далее будет выдавать ошибку.

{CACHE number | NOCACHE} – количество уже заготовленных уникальных значений.

При достижении которого sequence начинает генерировать новые.

Примеры:

CREATE SEQUENCE s1;

Получить следующее значение и текущие значения:

SELECT s1.nextval FROM dual;

SELECT s1.currval from dual;

Использование в таблицах:

INSERT INTO students (student\_id, name, last\_name)

VALUES s1.nexval, ‘TestName’, ‘TestLastName’;

Изменение SEQUENCE

ALTER SEQUENCE *schema.*sequence\_name

*INCREMENT BY number*

*{MAXVALUE number | NOMAXVALUE}*

*{MINVALUE number | NOMINVALUE}*

*{CYCLE | NOCYCLE}*

*{CACHE number | NOCACHE};*

Пример:

ALTER SEQUENCE s1 INCREMENT BY 5;

Удаление SEQUENCE

DROP SEQUENCE schema.sequence\_name;