Literature:

<https://drive.google.com/open?id=0B8WDvzG3rLBfTVR6cGpaNThOY1k>

http://www.slideshare.net/ssuser61084c/oracle-1-39573899

OracleScript:

<https://drive.google.com/file/d/0B8WDvzG3rLBfTG9WcUFqUHNoOVk/view?usp=sharing>

create TABLE customer

( customer\_id number(10) NOT NULL,

customer\_name varchar2(50) NOT NULL,

city varchar2(50)

);

ALTER table customer

Add customer\_phone varchar(45)

alter table customer

drop column city

alter table customer

add constraint customer\_PK primary key (customer\_id)

alter table customer

enable constraint customer\_PK

alter table customer

disable constraint customer\_PK

create TABLE order

(

order\_id number(10) NOT NULL,

order\_date date,

order\_name varchar2(50) NOT NULL

);

alter table order

add customer\_id\_FK number(10)

alter table order

add constraint order\_FK

foreign key (customer\_id\_FK)

references customer (customer\_id);

alter table order

add constraint order\_PK primary key (order\_id)

CREATE TABLE table\_name  
(   
 column1 datatype [ NULL | NOT NULL ],  
 **column2 datatype [ NULL | NOT NULL ],**  
 ...  
 column\_n datatype [ NULL | NOT NULL ]  
);

CREATE TABLE customers  
( customer\_id number(10) NOT NULL,  
 customer\_name varchar2(50) NOT NULL,  
 city varchar2(50)  
);

ALTER TABLE table\_name  
 ADD **column\_name column-definition;**

ALTER TABLE customers  
 ADD customer\_phone varchar2(45);

ALTER TABLE table\_name  
 DROP COLUMN city;

ALTER TABLE table\_name  
ADD CONSTRAINT constraint\_name PRIMARY KEY (column1, column2, ... column\_n);

ALTER TABLE customer   
ADD CONSTRAINT customer\_PK PRIMARY KEY (customer\_id );

## **Disable Primary Key**

## ALTER TABLE table\_name DISABLE CONSTRAINT constraint\_name;

## **Enable Primary Key**

ALTER TABLE table\_name  
ENABLE CONSTRAINT constraint\_name;

CREATE TABLE orders

( order\_id number(10) NOT NULL,

order\_date date

);

ALTER TABLE orders  
 ADD customer\_id\_fk number(10);

### 

ALTER TABLE table\_name  
ADD CONSTRAINT constraint\_name  
 FOREIGN KEY (column1, column2, ... column\_n)  
 REFERENCES parent\_table (column1, column2, ... column\_n);

ALTER TABLE orders  
ADD CONSTRAINT orders\_FK  
 FOREIGN KEY (customer\_id\_fk )  
 REFERENCES customers (customer\_id );

ALTER TABLE orders  
ADD CONSTRAINT orders\_PK  
 PRIMARY KEY (order\_id )

CREATE TABLE new\_table  
 AS (SELECT column\_name1, column\_name2….,column\_name\_n FROM old\_table);

select sin(pi) “my name” from dual

CREATE TABLE vendor  
 AS (SELECT customer\_id vendor\_id, customer\_name vendor\_name FROM customers);

# **Constraints**

# **Unique Constraints**

create TABLE customer

( customer\_id number(10) NOT NULL,

customer\_name varchar2(50) NOT NULL,

customer\_phone varchar2(50)

);

ALTER TABLE customers  
 ADD customer\_fax varchar2(50);

ALTER TABLE table\_name  
ADD CONSTRAINT constraint\_name UNIQUE (column1, column2, ... column\_n);

ALTER TABLE customers  
ADD CONSTRAINT phone\_UNIQUE UNIQUE (customer\_phone , customer\_fax);

## ALTER TABLE customers DISABLE CONSTRAINT phone\_UNIQUE ;

# **Check Constraints**

ALTER TABLE table\_name  
ADD CONSTRAINT constraint\_name CHECK (column\_name condition) ;

ALTER TABLE table\_name  
ADD CONSTRAINT constraint\_name CHECK (column\_name condition) [DISABLE];

ALTER TABLE customers  
ADD CONSTRAINT customer\_name\_CHECK  
 CHECK (customer\_name IN ('Ivan', 'Petro', 'Vova')) DISABLE;

# **Synonyms**

create TABLE aaaa

( product\_id number(10) NOT NULL,

product\_name varchar2(50) NOT NULL,

);

CREATE [OR REPLACE] [PUBLIC] SYNONYM [schema .] synonym\_name  
 FOR [schema .] object\_name [@ dblink];

CREATE PUBLIC SYNONYM product  
FOR app.aaaa;

# **DROP TABLE Statement**

DROP TABLE [schema\_name].table\_name  
[ CASCADE CONSTRAINTS ]  
[ PURGE ];

DROP TABLE customers CASCADE CONSTRAINTS;

<https://drive.google.com/file/d/0B8WDvzG3rLBfTG9WcUFqUHNoOVk/view?usp=sharing>

select 'a' || 'b' from dual

Использование метасимволов для фильтрации

Использование логического оператора LIKE

SELECT prod\_id, prod\_name

FROM Products

WHERE prod\_name LIKE 'Fish%';

SELECT prod\_id, prod\_name

FROM Products

WHERE prod\_name LIKE '%bean bag%';

"символ подчеркивания" (\_)

SELECT prod\_id, prod\_name

FROM Products

WHERE prod\_name LIKE '\_\_ inch teddy bear%';

SELECT prod\_id, prod\_name

FROM Products

WHERE prod\_name LIKE '% inch teddy bear%';

"квадратные скобки" ([ ])

SELECT cust\_contact

FROM Customers

WHERE cust\_contact LIKE '[JM]% '

ORDER BY cust\_contact

SELECT cust\_contact

FROM Customers

WHERE (cust\_contact LIKE 'J% ' ) OR( cust\_contact LIKE 'M% ' )

ORDER BY cust\_contact

# Date/Time Functions

SYSDATE

select SYSDATE from dual --Select 5 from dual

SYSTIMESTAMP

select SYSTIMESTAMP from dual

### **TO\_CHAR** Function

TO\_CHAR( value [, format\_mask] [, nls\_language] )

|  |  |
| --- | --- |
| **Parameter** | **Explanation** |
| YEAR | Year, spelled out |
| YYYY | 4-digit year |
| YYY  YY  Y | Last 3, 2, or 1 digit(s) of year. |
| IYY  IY  I | Last 3, 2, or 1 digit(s) of ISO year. |
| IYYY | 4-digit year based on the ISO standard |
| Q | Quarter of year (1, 2, 3, 4; JAN-MAR = 1). |
| MM | Month (01-12; JAN = 01). |
| MON | Abbreviated name of month. |
| MONTH | Name of month, padded with blanks to length of 9 characters. |
| RM | Roman numeral month (I-XII; JAN = I). |
| WW | Week of year (1-53) where week 1 starts on the first day of the year and continues to the seventh day of the year. |
| W | Week of month (1-5) where week 1 starts on the first day of the month and ends on the seventh. |
| IW | Week of year (1-52 or 1-53) based on the ISO standard. |
| D | Day of week (1-7). |
| DAY | Name of day. |
| DD | Day of month (1-31). |
| DDD | Day of year (1-366). |
| DY | Abbreviated name of day. |
| J | Julian day; the number of days since January 1, 4712 BC. |
| HH | Hour of day (1-12). |
| HH12 | Hour of day (1-12). |
| HH24 | Hour of day (0-23). |
| MI | Minute (0-59). |
| SS | Second (0-59). |
| SSSSS | Seconds past midnight (0-86399)., ‘’ |
| FF | Fractional seconds. |

SELECT TO\_CHAR(SYSTIMESTAMP, 'SSSS.FF')  
FROM dual;

### SYSDATE

*Result:* 'July 4, 2003'

SELECT TO\_CHAR(SYSDATE, 'MONTH D YYYY') FROM dual;

SELECT TO\_CHAR(SYSDATE, 'DDD') FROM dual;

ADD\_MONTHS( date1, number\_months )

SELECT ADD\_MONTHS('01-Aug-03', -3) FROM DUAL

2003-08-01

### EXTRACT function

EXTRACT (  
{ YEAR | MONTH | DAY | HOUR | MINUTE | SECOND }  
| { TIMEZONE\_HOUR | TIMEZONE\_MINUTE }  
| { TIMEZONE\_REGION | TIMEZONE\_ABBR }  
FROM { date\_value | interval\_value } )

select EXTRACT(YEAR FROM DATE '2003-08-22') from dual;

select EXTRACT(YEAR FROM SYSDATE) from dual;

select EXTRACT(DAY FROM SYSDATE) from dual;

NEXT\_DAY

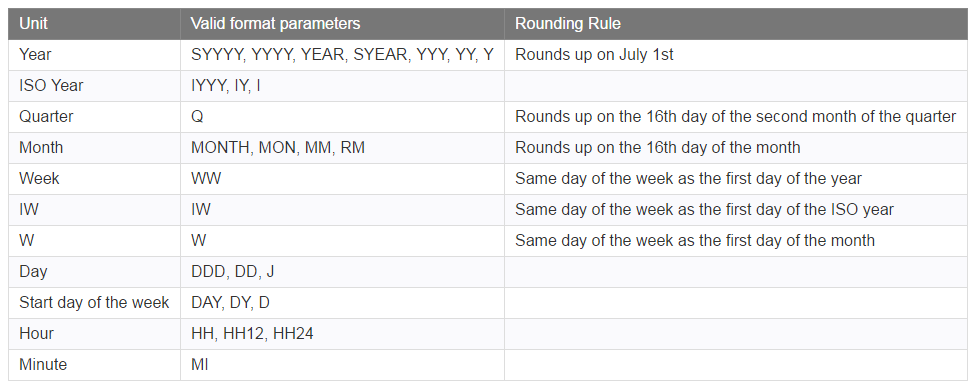
### NEXT\_DAY( date, weekday )

select NEXT\_DAY('9-Mar-17','FRIDAY') from dual;

<http://web.library.yale.edu/cataloging/months.htm>

### ROUND function

**ROUND( date [, format] )**



**ROUND(TO\_DATE ('22-AUG-03'),'YEAR')**

***Result:* '01-JAN-04'**

**select ROUND(TO\_DATE('2003/08/22', 'YYYY/MM/DD'),'YEAR') from dual;  
  
ROUND(TO\_DATE ('22-AUG-03'),'Q')  
*Result:* '01-OCT-03'  
  
ROUND(TO\_DATE ('22-AUG-03'),'MONTH')  
*Result:* '01-SEP-03'  
  
ROUND(TO\_DATE ('1-AUG-03'),'DDD')  
*Result:* '1-AUG-03'  
  
ROUND(TO\_DATE ('22-AUG-03'),'DAY')  
*Result:* '24-AUG-03'**

### TRUNC function

**ROUND(TO\_DATE ('22-AUG-03'),'YEAR')**

***Result:* '01-JAN-04'**

**select ROUND(TO\_DATE('2003/08/22', 'YYYY/MM/DD'),'YEAR') from dual;  
  
ROUND(TO\_DATE ('22-AUG-03'),'Q')  
*Result:* '01-OCT-03'  
  
ROUND(TO\_DATE ('22-AUG-03'),'MONTH')  
*Result:* '01-SEP-03'  
  
ROUND(TO\_DATE ('1-AUG-03'),'DDD')  
*Result:* '1-AUG-03'  
  
ROUND(TO\_DATE ('22-AUG-03'),'DAY')  
*Result:* '24-AUG-03'**

# Numeric/Math Functions

**AVG** function returns the average value of an expression.

SELECT AVG(aggregate\_expression)  
FROM tables

select avg(prod\_price) from products

# Numeric/Math Functions

**AVG** function returns the average value of an expression.

SELECT AVG(aggregate\_expression)  
FROM tables

select avg(prod\_price) from products

**CEIL** function returns the smallest integer value that is greater than or equal to a *number*.

CEIL( number )

select CEIL(32.65) from dual;

select CEIL(32.1) from dual

select CEIL(-32.65) from dual

**FLOOR** function returns the largest integer value that is equal to or less than a *number*.

select FLOOR(5.9) from dual

*Result:* 5

select FLOOR(-5.9) from dual

*Result:* -6

**GREATEST** function returns the greatest value in a list of expressions.

select GREATEST(2, 5, 12, 3) from dual

GREATEST( expr1 [, expr2, ... expr\_n] )

select GREATEST('apples', 'oranges', 'bananas') from dual

GREATEST('apples', 'applis', 'applas')

**LEAST** function returns the smallest value in a list of expressions.

LEAST( expr1 [, expr2, ... expr\_n] )

select GREATEST('apples', 'applis', 'applas') from dual

**ROUND** function returns a number rounded to a certain number of decimal places.

ROUND( number [, decimal\_places] )

ROUND(125.315)  
*Result:* 125  
  
ROUND(125.315, 0)  
*Result:* 125  
  
ROUND(125.315, 1)  
*Result:* 125.3  
  
ROUND(125.315, 2)  
*Result:* 125.32  
  
ROUND(125.315, 3)  
*Result:* 125.315

**TRUNC** function returns a number truncated to a certain number of decimal places.

TRUNC( number [, decimal\_places] )

TRUNC(125.815)  
*Result:* 125  
  
TRUNC(125.815, 0)  
*Result:* 125  
  
TRUNC(125.815, 1)  
*Result:* 125.8  
  
TRUNC(125.815, 2)  
*Result:* 125.81  
  
TRUNC(125.815, 3)  
*Result:* 125.815  
  
TRUNC(-125.815, 2)  
*Result:* -125.81  
  
TRUNC(125.815, -1)  
*Result:* 120  
  
TRUNC(125.815, -2)  
*Result:* 100  
  
TRUNC(125.815, -3)  
*Result:* 0

select \* from customers

where cust\_name LIKE 'V%'; --\_ [] %

select \* from customers

where REGEXP\_LIKE ( expression, pattern [, match\_parameter ] );

|  |  |
| --- | --- |
| ^ | Matches the beginning of a string. If used with a *match\_parameter* of 'm', it matches the start of a line anywhere within *expression*. |
| $ | Matches the end of a string. If used with a *match\_parameter* of 'm', it matches the end of a line anywhere within *expression*. |
| \* | Matches zero or more occurrences. |
| + | Matches one or more occurrences. |
| ? | Matches zero or one occurrence. |
| . | Matches any character except NULL. |
| | | Used like an "OR" to specify more than one alternative. |
| [ ] | Used to specify a matching list where you are trying to match any one of the characters in the list. |
| [^ ] | Used to specify a nonmatching list where you are trying to match any character except for the ones in the list. |
| ( ) | Used to group expressions as a subexpression. |
| {m} | Matches m times. |
| {m,} | Matches at least m times. |
| {m,n} | Matches at least m times, but no more than n times. |
| \n | n is a number between 1 and 9. Matches the nth subexpression found within ( ) before encountering \n. |
| [..] | Matches one collation element that can be more than one character. |
| [::] | Matches character classes. |
| [==] | Matches equivalence classes. |
| \d | Matches a digit character. |
| \D | Matches a nondigit character. |
| \w | Matches a word character. |
| \W | Matches a nonword character. |
| \s | Matches a whitespace character. |
| \S | matches a non-whitespace character. |
| \A | Matches the beginning of a string or matches at the end of a string before a newline character. |
| \Z | Matches at the end of a string. |
| \*? | Matches the preceding pattern zero or more occurrences. |
| +? | Matches the preceding pattern one or more occurrences. |
| ?? | Matches the preceding pattern zero or one occurrence. |
| {n}? | Matches the preceding pattern n times. |
| {n,}? | Matches the preceding pattern at least n times. |
| {n,m}? | Matches the preceding pattern at least n times, but not more than m times. |

Адрес, который содержит от 3-4 цифр в начале.

# **Check Constraints**

CREATE TABLE table\_name  
(  
 column1 datatype null/not null,  
 column2 datatype null/not null,  
  
 ...  
  
 CONSTRAINT constraint\_name CHECK (column\_name condition) [DISABLE]  
  
);

ALTER TABLE table\_name  
ADD CONSTRAINT constraint\_name CHECK (column\_name condition) [DISABLE];

CREATE TABLE PAYMENTS

(

PAYMENT\_DATA DATE,

PAYMENT\_CUSTOMER CHAR(10 byte),

PAYMENT\_NUMBER CHAR(50)

);

xxx.xxx.xxx.xxx

(\d{3}\.){3}\d{3}

ALTER TABLE PAYMENTS

ADD CONSTRAINT payment\_number\_CHECK CHECK (REGEXP\_LIKE(PAYMENT\_NUMBER , ’(\d{3}\.){3}\d{3}’) ) DISABLE;

**ALTER TABLE PAYMENTS**

**ADD CONSTRAINT payment\_customer\_CHECK**

**CHECK (REGEXP\_LIKE(PAYMENT\_CUSTOMER, '^[A-Z][a-z]\*\s[A-Z]\.\s[A-Z]\.\s\*'));**

**ALTER TABLE PAYMENTS**

**DROP CONSTRAINT payment\_customer\_CHECK;**

**ALTER TABLE PAYMENTS**

**ADD CONSTRAINT payment\_customer\_CHECK**

**CHECK (REGEXP\_LIKE(PAYMENT\_CUSTOMER , '^[\u0410-\u042F]\w+\s[A-Z]\.\s[A-Z]\.\s\*')) DISABLE;**

**[a-zA-Z\u0410-\u042F\u0430-\u044F\u0401\u0451\u0101\u0100\u010c\u010d\u0112\u0113\u011E\u011F\u012A\u012B\u0136\u0137\u013b\u013C\u0145\u0146\u0160\u0161\u016A\u016B\u017D\u017E]$**

**https://m.unicode-table.com/hu/alphabets/ukrainian/**

[**https://www.terena.org/activities/multiling/koi8-ru/unicode0400-ru.html**](https://www.terena.org/activities/multiling/koi8-ru/unicode0400-ru.html)

**SELECT PRODUCTS.PROD\_NAME, CUSTOMERS.CUST\_NAME**

**FROM CUSTOMERS,ORDERS,ORDERITEMS,PRODUCTS**

**WHERE**

**CUSTOMERS.CUST\_ID=ORDERS.CUST\_ID and**

**ORDERS.ORDER\_NUM=ORDERITEMS.ORDER\_NUM AND**

**PRODUCTS.PROD\_ID = ORDERITEMS.PROD\_ID;**

**SELECT**

**PRODUCTS.PROD\_NAME,**

**CUSTOMERS.CUST\_NAME**

**FROM**

**CUSTOMERS JOIN ORDERS**

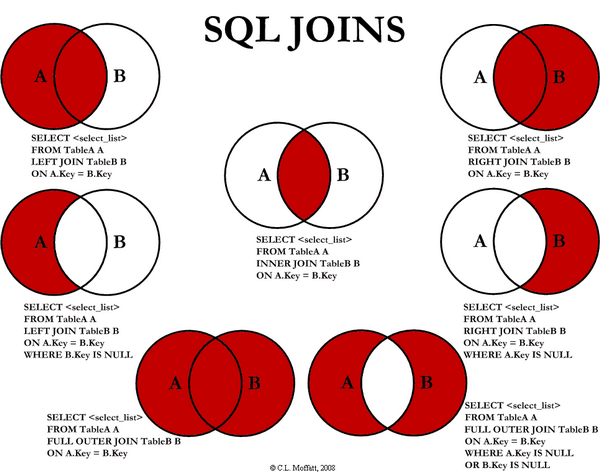
**ON CUSTOMERS.CUST\_ID=ORDERS.CUST\_ID**

**JOIN ORDERITEMS**

**ON ORDERS.ORDER\_NUM=ORDERITEMS.ORDER\_NUM**

**JOIN PRODUCTS**

**ON PRODUCTS.PROD\_ID = ORDERITEMS.PROD\_ID;**



**-- Вывести всех кастомеров и указать у каких вендоров они ничего не покупали**

**SELECT ORDERS.CUST\_ID "Customer", VENDORS.VEND\_ID "Vendor"**

**FROM ORDERS**

**JOIN ORDERITEMS**

**ON ORDERITEMS.ORDER\_NUM = ORDERS.ORDER\_NUM**

**JOIN PRODUCTS**

**ON ORDERITEMS.PROD\_ID = PRODUCTS.PROD\_ID**

**JOIN VENDORS**

**ON PRODUCTS.VEND\_ID = VENDORS.VEND\_ID;**

**SELECT CUSTOMERS.CUST\_ID "Customer", VENDORS.VEND\_ID "Vendor"**

**FROM CUSTOMERS, VENDORS**

**MINUS**

**SELECT ORDERS.CUST\_ID "Customer", VENDORS.VEND\_ID "Vendor"**

**FROM ORDERS JOIN ORDERITEMS**

**ON ORDERITEMS.ORDER\_NUM = ORDERS.ORDER\_NUM**

**JOIN PRODUCTS**

**ON ORDERITEMS.PROD\_ID = PRODUCTS.PROD\_ID**

**JOIN VENDORS**

**ON PRODUCTS.VEND\_ID = VENDORS.VEND\_ID;**

**-------------------------------**

**SELECT CUSTOMERS.CUST\_NAME, VENDORS.VEND\_NAME**

**FROM**

**(SELECT CUSTOMERS.CUST\_ID "Customer", VENDORS.VEND\_ID "Vendor"**

**FROM CUSTOMERS, VENDORS**

**MINUS**

**SELECT ORDERS.CUST\_ID "Customer", VENDORS.VEND\_ID "Vendor"**

**FROM ORDERS**

**JOIN ORDERITEMS**

**ON ORDERITEMS.ORDER\_NUM = ORDERS.ORDER\_NUM**

**JOIN PRODUCTS**

**ON ORDERITEMS.PROD\_ID = PRODUCTS.PROD\_ID**

**JOIN VENDORS**

**ON PRODUCTS.VEND\_ID = VENDORS.VEND\_ID) filter**

**JOIN CUSTOMERS**

**ON CUSTOMERS.CUST\_ID = filter."Customer"**

**JOIN VENDORS**

**ON VENDORS.VEND\_ID = filter."Vendor";**

**------------------------------------------------------**

**select cast(CUSTOMERS.CUST\_NAME as char(50)) Human from CUSTOMERS**

**UNION all**

**select cast(VENDORS.VEND\_NAME as char(50)) Human from VENDORS;**

SELECT

CUSTOMERS.CUST\_ID,

CUSTOMERS.CUST\_NAME,

customers\_copy.CUST\_ID,

customers\_copy.CUST\_NAME

FROM

CUSTOMERS

JOIN

CUSTOMERS customers\_copy

ON (CUSTOMERS.CUST\_NAME=customers\_copy.CUST\_NAME

and

CUSTOMERS.CUST\_ID!=customers\_copy.CUST\_ID)

PL SQL

**DECLARE**

**Price NUMBER(5,2) := 12.43; *-- Переменная типа NUMBER (с плавающей точкой***

**BEGIN**

DBMS\_OUTPUT.put\_line(Price);

**END;**

SET SERVEROUTPUT ON

BEGIN

DBMS\_OUTPUT.enable;

DBMS\_OUTPUT.put\_line(‘Hello !’);

END**;**

**DECLARE**

**Price NUMBER(5,2) := 12.43; *-- Переменная типа NUMBER (с плавающей точкой***

**BEGIN**

DBMS\_OUTPUT.put\_line(Price);

**END;**

**DECLARE**

**count\_of\_customers INTEGER:=0;**

**BEGIN**

**select count(\*) into count\_of\_customers from customers;**

DBMS\_OUTPUT.put\_line(**count\_of\_customers** );

**END;**

**DECLARE**

**Price NUMBER(5,2) := 12.43; -- Переменная типа NUMBER (с плавающей точкой**

**CCount int := 0;**

**CustID char(10) := '';**

**CustName char(50) := '';**

**BEGIN**

**DBMS\_OUTPUT.put\_line(Price);**

**select count(\*) into CCount from CUSTOMERS;**

**DBMS\_OUTPUT.put\_line(CCount);**

**select CUST\_ID, CUST\_NAME into CustID, CustName from CUSTOMERS where CUST\_ID = '1000000001';**

**DBMS\_OUTPUT.put\_line(CustID || ' ' || CustName);**

**END;**

**IF ( val = 1 ) THEN *-- проверка условия***  
 .  
 .  
ELSIF ( val = 2 ) THEN   
 .  
 .  
ELSIF ( val = 3 ) THEN   
 .  
 .  
ELSIF ( val = 9 ) THEN   
 .  
 .  
ELSE *-- не сработало не одно из условий!*

.  
 .  
  
  
**END IF; *-- конец условного оператора.***

DECLARE

count\_of\_customers INTEGER := 0;

BEGIN

SELECT

COUNT(\*) INTO count\_of\_customers

FROM

CUSTOMERS ;

IF(count\_of\_customers = 0) THEN

DBMS\_OUTPUT.PUT\_LINE('NO customers');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Count of customers: ' || count\_of\_customers);

END IF;

END;

LOOP

NULL;

END LOOP

1. **EXIT** - Безусловный выход из цикла. Используется посредством применения оператора **IF**.
2. **EXIT WHEN** - Выход при выполнении условия.
3. **GOTO** - Выход из цикла во внешний контекст.

DECLARE  
 i NUMBER := 0;

**BEGIN**

**i:=0;**

**LOOP**

**DBMS\_OUTPUT.PUT\_LINE(i);**

**i:=i+1;**

**IF(i>100) THEN**

**EXIT;**

**END IF  
END LOOP**

**i:=0;**

**LOOP**

**DBMS\_OUTPUT.PUT\_LINE(i);**

**i:=i+1;**

**EXIT WHEN(i>100)  
END LOOP**

**i:=0;**

**WHILE (i<100) LOOP**

**DBMS\_OUTPUT.PUT\_LINE(i);**

**i:=i+1;**

**END LOOP**

**END**

DECLARE

**CURSOR** CUST\_NAME\_CURSOR

**IS**

SELECT

CUST\_NAME, CUST\_ID

FROM

CUSTOMERS**;**

BEGIN

FOR record IN CUST\_NAME\_CURSOR

LOOP

DBMS\_OUTPUT.put\_line(record .CUST\_NAME || ‘ ‘ ||record .CUST\_ID);

END LOOP;

END;

**cust id**

**cust\_name**

**type**

-VIP >1000

-MID <1000 >400

-COMMON else

DECLARE

cust\_type CHAR(50) := 'not known';

CURSOR test\_cursor

IS

SELECT customers.cust\_name, customers.cust\_id, SUM(orderitems.quantity\*orderitems.item\_price) orders\_sum FROM customers

LEFT JOIN orders

ON customers.cust\_id = orders.cust\_id

LEFT JOIN orderitems

ON orders.order\_num = orderitems.order\_num

GROUP BY customers.cust\_name, customers.cust\_id;

BEGIN

FOR rec IN test\_cursor

LOOP

IF(rec.orders\_sum > 1000) THEN

cust\_type := 'VIP';

ELSIF(rec.orders\_sum > 400) THEN

cust\_type := 'MID';

ELSE

cust\_type := 'COMMON';

END IF;

DBMS\_OUTPUT.put\_line(rec.cust\_id || ' ' || rec.cust\_name || cust\_type);

END LOOP;

END;

DECLARE

**var1 CUSTOMERS.CUST\_ID%TYPE;**

**var2 CUSTOMERS.CUST\_NAME%TYPE;**

**var3 CUSTOMERS%ROWTYPE;**

**CURSOR** CUST\_NAME\_CURSOR

**IS**

SELECT

CUST\_NAME, CUST\_ID

FROM

CUSTOMERS**;**

BEGIN

FOR record IN CUST\_NAME\_CURSOR

LOOP

var1:=record.CUST\_ID;

var2:=record.CUST\_NAME;

var3:=record;

END LOOP;

END;

**DECLARE**

**CUST\_NAME\_DATA CUSTOMERS.CUST\_NAME%TYPE;**

**CUST\_ID\_DATA CUSTOMERS.CUST\_ID%TYPE;**

CURSOR CUSTOMERS\_CURSOR (**customer\_name\_filter CUSTOMERS.CUST\_NAME%TYPE**)

IS

**SELECT**

**CUST\_NAME,**

**CUST\_ID**

**FROM**

**CUSTOMERS**

**WHERE**

**CUSTOMERS.CUST\_NAME=customer\_name\_filter ;**

**BEGIN**

**OPEN CUSTOMERS\_CURSOR( 'Fun4All');**

**LOOP**

**FETCH CUSTOMERS\_CURSOR INTO CUST\_NAME\_DATA ,CUST\_ID\_DATA ;**

**IF (CUSTOMERS\_CURSOR%FOUND) THEN**

**DBMS\_OUTPUT.PUT\_LINE(‘TRUE’);**

**DBMS\_OUTPUT.PUT\_LINE('CUST\_NAME: ' || CUST\_NAME\_DATA || ' CUST\_ID: ' || CUST\_ID\_DATA);**

**ELSE**

**DBMS\_OUTPUT.PUT\_LINE(‘FALSE’);**

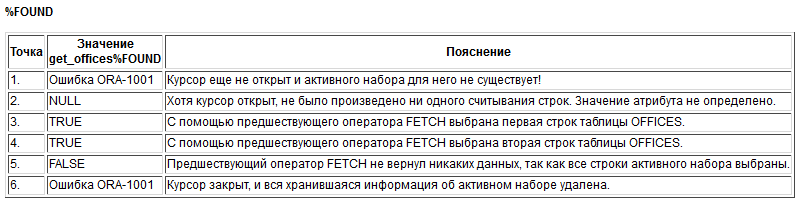
**EXIT;**

**END IF;**

**END LOOP;**

**CLOSE CUSTOMERS\_CURSOR;**

**END;**



**-------------------------------------------------------------------------------**

**-------------------------------------------------------------------------------**

**DECLARE**

**CURSOR CUSTOMERS\_CURSOR (customer\_name\_filter CUSTOMERS.CUST\_COUNTRY%TYPE)**

**IS**

**SELECT CUST\_ID, CUST\_NAME FROM CUSTOMERS**

**WHERE CUST\_COUNTRY =customer\_name\_filter;**

**CURSOR CUSTOMER\_ORDERS\_CURSOR (customer\_order\_filter ORDERS.CUST\_ID%TYPE)**

**IS**

**SELECT ORDER\_NUM FROM ORDERS**

**WHERE CUST\_ID =customer\_order\_filter;**

**CUST\_NAME\_DATA CUSTOMERS.CUST\_NAME%TYPE;**

**CUST\_ID\_DATA CUSTOMERS.CUST\_ID%TYPE;**

**ORDER\_NUM ORDERS.ORDER\_NUM%TYPE;**

**i int:=1;**

**BEGIN**

**OPEN CUSTOMERS\_CURSOR('USA');**

**loop**

**DBMS\_OUTPUT.PUT\_LINE('--------------------------------------------------');**

**i:=1;**

**FETCH CUSTOMERS\_CURSOR INTO CUST\_ID\_DATA, CUST\_NAME\_DATA;**

**IF (CUSTOMERS\_CURSOR%FOUND) THEN**

**DBMS\_OUTPUT.PUT\_LINE('CUST\_NAME: ' || CUST\_NAME\_DATA);**

**--DBMS\_OUTPUT.PUT\_LINE('--------------------------------------------------');**

**OPEN CUSTOMER\_ORDERS\_CURSOR(CUST\_ID\_DATA);**

**loop**

**FETCH CUSTOMER\_ORDERS\_CURSOR INTO ORDER\_NUM;**

**if (CUSTOMER\_ORDERS\_CURSOR%FOUND) THEN**

**DBMS\_OUTPUT.PUT\_LINE(SYSDATE ||' '|| i || ' ' || ORDER\_NUM);**

**i:=i+1;**

**else**

**exit;**

**end if;**

**end loop;**

**close CUSTOMER\_ORDERS\_CURSOR;**

**else**

**exit;**

**end if;**

**end loop;**

**close CUSTOMERS\_CURSOR;**

**END;**

**EXECUTE IMMEDIATE dynamic\_string  
[INTO {define\_variable[, define\_variable]... | record}]  
[USING [IN | OUT | IN OUT] bind\_argument  
 [, [IN | OUT | IN OUT] bind\_argument]...]  
[{RETURNING | RETURN} INTO bind\_argument[, bind\_argument]...];**

**begin  
 execute immediate 'whatever\_text\_string\_you\_want';  
end;**

**declare**

**var\_1 char(30):='SELECT \* FROM';**

**var\_2 char(30):=' CUSTOMERS';**

**begin**

**execute immediate var\_1||var\_2 ;**

**end;**

**declare**

**CUST\_NAME\_DATA char(30);**

**query\_str char(200);**

**begin**

**query\_str:=' SELECT '' '|| CUST\_NAME\_DATA ||' '', VENDORS.VEND\_NAME FROM VENDORS; ';**

**DBMS\_OUTPUT.put\_line(query\_str);**

**query\_str:=' SELECT '' '|| CUST\_NAME\_DATA ||' '',ORDER\_NUM FROM ORDERS JOIN CUSTOMERS ON CUSTOMERS.CUST\_ID=ORDERS.CUST\_ID and CUSTOMERS.CUST\_NAME= '''||CUST\_NAME\_DATA||'''';**

**DBMS\_OUTPUT.put\_line(query\_str);**

**end;**

**Вывод CUST\_NAME VENDOR\_NAME если CUST\_EMAIL IS NULL ИНАЧЕ выводим CUST\_NAME ORDER\_NUM**

**SET SERVEROUTPUT ON;**

**declare**

**vCUST\_ID CUSTOMERS.CUST\_ID%TYPE;**

**vCUST\_EMAIL CUSTOMERS.CUST\_EMAIL%TYPE;**

**vSQL varchar(255);**

**cursor customer\_cursor is select CUST\_ID, CUST\_EMAIL from CUSTOMERS;**

**begin**

**open customer\_cursor;**

**loop**

**fetch customer\_cursor into vCUST\_ID, vCUST\_EMAIL;**

**IF (not customer\_cursor%found) THEN**

**exit;**

**end if;**

**if (vCUST\_EMAIL is null) then**

**vSQL:= 'select CUST\_NAME, VEND\_NAME from CUSTOMERS, VENDORS where CUST\_ID = ' || vCUST\_ID ||';';**

**else**

**vSQL:= 'select c.CUST\_NAME, ORDER\_NUM from CUSTOMERS c inner join ORDERS o on c.CUST\_ID = o.CUST\_ID where c.CUST\_ID = ' || vCUST\_ID ||';';**

**end if;**

**DBMS\_OUTPUT.PUT\_LINE(vSQL);**

**end loop;**

**close customer\_cursor;**

**end;**

**DECLARE**

**query\_str char(300);**

**var1 CUSTOMERS.CUST\_NAME%TYPE;**

**var2 CUSTOMERS.CUST\_NAME%TYPE;**

**var3 CUSTOMERS.CUST\_NAME%TYPE;**

**customer\_email CUSTOMERS.CUST\_EMAIL%TYPE;**

**BEGIN**

**query\_str:=' SELECT :email1, :email1, :email1 FROM DUAL';**

**customer\_email:='a';**

**EXECUTE IMMEDIATE query\_str INTO var1,var2,var3 USING customer\_email,'b','c';**

**DBMS\_OUTPUT.put\_line(var1||var2||var3);**

**END;**

**Select ? from customers where cust\_id = ?**

**declare s varchar(255);**

**cId CUSTOMERS.CUST\_ID%type:= '1000000001';**

**cName CUSTOMERS.CUST\_NAME%type;**

**cFieldName varchar(255):= 'CUST\_NAME';**

**begin**

**s:='select '||cFieldName||' from customers where cust\_id = :p2';**

**execute immediate s into cName using cId;**

**DBMS\_OUTPUT.PUT\_LINE(cName);**

**end;**

**TYPE curs\_type IS REF CURSOR**

**cur\_variable curs\_type ;**

**OPEN cur\_variable FOR 'SELECT ...'**

**LOOP**

**END LOOP**

**DECLARE**

**TYPE customer\_cursor\_type IS REF CURSOR;**

**customer\_cursor customer\_cursor\_type;**

**query\_str char(300);**

**begin**

**query\_str :=’Select CUST\_NAME from CUSTOMERS’;**

**OPEN customer\_cursor FOR query\_str**

**LOOP**

**FETCH customer\_cursor into ….**

**exit when (customer\_cursor %NOTFOUND);**

**END LOOP**

**end;**

**DECLARE**

**TYPE customers\_cursor\_type IS REF CURSOR;**

**customers\_cursor customers\_cursor\_type;**

**customer\_name customers.cust\_name%type;**

**query\_str varchar(300);**

**BEGIN**

**query\_str :='SELECT cust\_name FROM customers';**

**OPEN customers\_cursor FOR query\_str;**

**LOOP**

**FETCH customers\_cursor INTO customer\_name;**

**EXIT WHEN (customers\_cursor%NOTFOUND);**

**DBMS\_OUTPUT.put\_line(customer\_name);**

**END LOOP;**

**END;**

Вывести количество Customers имя которых удовлетворяет регулярному выражению

**DECLARE**

**query\_str varchar(500);**

**cust\_name\_pattern varchar(500) := '^[A-Z][a-z]+\s[A-Z][a-z]+\s\*$';**

**res integer;**

**BEGIN**

**query\_str := 'SELECT COUNT(customers.cust\_name) FROM customers WHERE REGEXP\_LIKE(customers.cust\_name, :cust\_name\_pattern)';**

**EXECUTE IMMEDIATE query\_str INTO res USING cust\_name\_pattern;**

**DBMS\_OUTPUT.put\_line(res);**

**END;**

# **Hierarchical Queries**

Общий алгоритм вычисления фразой WITH таков:  
  
Результат := пусто;

Init\_query:= исходный SELECT ...;

Пока Init\_query не пуст выполнять:  
 Результат := Результат  
{UNION ALL | UNION | INTERSECT | EXCEPT}  
init\_query;  
 init\_query:= рекурсивный SELECT ... FROM init\_query…;  
конец цикла;

WITH

numbers ( n ) AS

(

SELECT 1 AS n FROM dual -- исходное множество -- одна строка

UNION ALL -- символическое «объединение» строк

SELECT n + 1 AS n -- рекурсия: добавок к предыдущему результату

FROM numbers -- предыдущий результат в качестве источника данных

WHERE n < 5 -- если не ограничить, будет бесконечная рекурсия

)

SELECT n FROM numbers -- основной запрос

CREATE TABLE tab1 (  
 id NUMBER,  
 parent\_id NUMBER,  
 CONSTRAINT tab1\_pk PRIMARY KEY (id),  
 CONSTRAINT tab1\_tab1\_fk FOREIGN KEY (parent\_id) REFERENCES tab1(id)  
);

INSERT INTO tab1 VALUES (1, NULL);  
INSERT INTO tab1 VALUES (2, 1);  
INSERT INTO tab1 VALUES (3, 2);  
INSERT INTO tab1 VALUES (4, 2);  
INSERT INTO tab1 VALUES (5, 4);  
INSERT INTO tab1 VALUES (6, 4);  
INSERT INTO tab1 VALUES (7, 1);  
INSERT INTO tab1 VALUES (8, 7);  
INSERT INTO tab1 VALUES (9, 1);  
INSERT INTO tab1 VALUES (10, 9);  
INSERT INTO tab1 VALUES (11, 10);  
INSERT INTO tab1 VALUES (12, 9);  
COMMIT;

WITH

table\_with\_current\_id(id, parent\_id)

AS

(

-- Anchor member.

SELECT

id,parent\_id --current id

FROM tab1  
 WHERE parent\_id IS NULL

-- Recursive member

UNION ALL

-- for current id

SELECT temp.id,  
 temp.parent\_id  
 FROM tab1 temp, table\_with\_current\_id

WHERE temp.parent\_id = table\_with\_current\_id.id

)

SELECT

id,  
 parent\_id  
FROM table\_with\_current\_id

WITH recursion\_table(id, parent\_id) AS (  
 -- Anchor member.  
 **SELECT id,** parent\_id  
 **FROM tab1  
 WHERE parent\_id IS NULL**

UNION ALL  
 -- Recursive member.  
 SELECT t2.id,  
 t2.parent\_id  
 FROM tab1 t2, recursion\_table  
 WHERE t2.parent\_id = recursion\_table.id  
)  
SELECT id,  
 parent\_id  
FROM recursion\_table;

* BREADTH FIRST BY : Sibling rows are returned before child rows are processed.

WITH t1(id, parent\_id) AS (  
 -- Anchor member.  
 SELECT id,  
 parent\_id  
 FROM tab1  
 WHERE parent\_id IS NULL  
 UNION ALL  
 -- Recursive member.  
 SELECT t2.id,  
 t2.parent\_id  
 FROM tab1 t2, t1  
 WHERE t2.parent\_id = t1.id  
)  
**SEARCH BREADTH FIRST BY id SET order1**SELECT id,  
 parent\_id  
FROM t1  
ORDER BY order1;

* DEPTH FIRST BY : Child rows are returned before siblings are processed.

WITH t1(id, parent\_id) AS (  
 -- Anchor member.  
 SELECT id,  
 parent\_id  
 FROM tab1  
 WHERE parent\_id IS NULL  
 UNION ALL  
 -- Recursive member.  
 SELECT t2.id,  
 t2.parent\_id  
 FROM tab1 t2, t1  
 WHERE t2.parent\_id = t1.id  
)  
SEARCH DEPTH FIRST BY id SET order1  
SELECT id,  
 parent\_id  
FROM t1  
ORDER BY order1;

WITH recursion\_table(id, parent\_id,level1) AS (  
 -- Anchor member.  
 **SELECT id,** parent\_id,

1 level1  
 **FROM tab1  
 WHERE parent\_id IS NULL**

UNION ALL  
 -- Recursive member.  
 SELECT t2.id,  
 t2.parent\_id,

level1+1  
 FROM tab1 t2, recursion\_table  
 WHERE t2.parent\_id = recursion\_table.id  
)

SEARCH DEPTH FIRST BY id SET order1

SELECT id,  
 level1  
FROM recursion\_table

ORDER BY order1;

WITH recursion\_table(id, parent\_id, lev, root) AS (

-- Anchor member.

SELECT id,

parent\_id,

1 lev,

TO\_CHAR(id) root

FROM tab1

WHERE parent\_id IS NULL

UNION ALL

-- Recursive member.

SELECT t2.id,

t2.parent\_id,

lev + 1,

root || '-' || t2.id

FROM tab1 t2, recursion\_table

WHERE t2.parent\_id = recursion\_table.id

)

SEARCH DEPTH FIRST BY id SET order1

SELECT id,

lev,

root

FROM recursion\_table

ORDER BY order1;

CREATE [OR REPLACE] PROCEDURE procedure\_name  
 [ (parameter [,parameter]) ]  
  
IS  
 [declaration\_section]  
  
BEGIN  
 executable\_section  
  
[EXCEPTION  
 exception\_section]  
  
END [procedure\_name];

CREATE OR REPLACE PROCEDURE HELLO( user\_name **IN** varchar2, prompt **IN** VARCHAR2)

is

BEGIN

DBMS\_OUTPUT.put\_line(prompt ||’ ’||user\_name );

END HELLO;

SET SERVEROUTPUT ON;

EXEC HELLO('User',’Hello’);

CREATE OR REPLACE PROCEDURE HELLO( user\_name **IN** varchar2, prompt\_param **IN** VARCHAR2 ***:='NELLO'***, text\_str **OUT** varchar2)

is

BEGIN

text\_str :=prompt\_param||' '|| user\_name;

DBMS\_OUTPUT.put\_line(text\_str );

END HELLO;

SET SERVEROUTPUT ON;

DECLARE

*USER\_NAME* VARCHAR2(200);

*PROMPT\_PARAM* VARCHAR2(200);

TEXT\_STR VARCHAR2(200);

BEGIN

USER\_NAME := 'NULL';

PROMPT\_PARAM := 'NULL';

HELLO(USER\_NAME ,PROMPT\_PARAM ,TEXT\_STR );

HELLO(

*USER\_NAME* => USER\_NAME,

*PROMPT\_PARAM* => PROMPT\_PARAM,

*TEXT\_STR* => TEXT\_STR

);

DBMS\_OUTPUT.put\_line('TEXT\_STR = ' ||TEXT\_STR );

END

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

create or replace PROCEDURE GetCustomerNameById(id IN CUSTOMERS.CUST\_ID%TYPE, text\_str OUT varchar2)

is

BEGIN

select CUST\_NAME into text\_str from CUSTOMERS where CUST\_ID = id;

END GetCustomerNameById;

----

DECLARE

ID CHAR(10);

TEXT\_STR VARCHAR2(200);

BEGIN

ID := '1000000001';

GETCUSTOMERNAMEBYID(

ID => ID,

TEXT\_STR => TEXT\_STR

);

DBMS\_OUTPUT.PUT\_LINE('TEXT\_STR = ' || TEXT\_STR);

END;

-------------- CREATE [OR REPLACE] FUNCTION - имя\_функции -------------------------------------  
-------------- (аргумент [IN] [OUT] [IN OUT] тип, ..... ) AS [IS] -----------------------------  
-------------- тело процедуры -----------------------------------------------------------------  
-------------- RETURN (возвращаемое\_значение) -------------------------------------------------

CREATE OR REPLACE FUNCTION count\_of\_customers(filter CUSTOMERS.CUST\_NAME%TYPE)

**RETURN INT**

AS

var INT;

BEGIN

SELECT COUNT(\*) INTO var FROM CUSTOMERS where cust\_name=filter;

**RETURN** var ;

END count\_of\_customers;

SELECT 2+4 FROM dual

select count\_of\_customers(“asfd”) from dual ;

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

CREATE OR REPLACE FUNCTION get\_order\_count\_by\_customer\_id(filter ORDERS.CUST\_ID%TYPE)

RETURN INT

AS

outCount INT;

BEGIN

SELECT COUNT(\*) INTO outCount FROM ORDERS where CUST\_ID=filter;

RETURN outCount;

END get\_order\_count\_by\_customer\_id;

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

select get\_order\_count\_by\_customer\_id('1000000001') from dual;

CREATE OR REPLACE PACKAGE имя\_модуля {IS AS}  
 описание\_процедуры |  
 описание\_функции |  
 объявление\_переменной |  
 определение\_типа |  
 объявление\_исключительной\_ситуации |  
 объявление\_курсора |  
END [имя\_модуля];

CREATE OR REPLACE PACKAGE **BODY** имя\_модуля {IS AS}  
 код\_инициализации\_процедуры |  
 код\_инициализации\_функции |  
END [имя\_модуля];

create or replace package test\_package is

function min(var1 number, var2 number) return number;

function max(var1 number, var2 number) return number;

end test\_package;

create or replace package body test\_package is

function min(var1 number, var2 number) return number as

begin

return LEAST(var1, var2);

end;

function max(var1 number, var2 number) return number as

begin

return GREATEST(var1, var2);

end;

end test\_package;

select test\_package.min(2,3) from dual

CREATE OR REPLACE FUNCTION count\_of\_customers(filter CUSTOMERS.CUST\_NAME%TYPE)

RETURN INT

-- row type

CREATE TYPE **customer\_row** AS OBJECT (

cust\_name char(50),

cust\_id char(10)

);

CREATE TYPE **customers\_table\_info** IS TABLE OF **customer\_row**;

CREATE OR REPLACE FUNCTION GetCustomers

RETURN **customers\_table\_info**

AS

result\_table **customers\_table\_info** := customers\_table\_info();

BEGIN

for cursor\_name in ( SELECT cust\_name, cust\_id from customers)

LOOP

--result\_table .addrow(cursor\_name )

result\_table.extend;

result\_table(result\_table.last):=**customer\_row**(cursor\_name.cust\_name,cursor\_name .cust\_id);

END LOOP;

RETURN result\_table ;

END GetCustomers;

SELECT \*

FROM TABLE(GetCustomers)

CREATE TYPE vendor\_row AS OBJECT (

vend\_name char(300),

prod\_name char(300)

);

CREATE TYPE vendors\_table\_info IS TABLE OF vendor\_row;

CREATE OR REPLACE FUNCTION GetVendors

RETURN vendors\_table\_info

AS

result\_table vendors\_table\_info := vendors\_table\_info();

BEGIN

FOR cursor\_name IN (SELECT vendors.vend\_name, products.prod\_name from vendors left join products on vendors.vend\_id = products.vend\_id)

LOOP

result\_table.extend;

result\_table(result\_table.last):=vendor\_row(cursor\_name.vend\_name, cursor\_name.prod\_name);

END LOOP;

RETURN result\_table;

END GetVendors;

SELECT \*

FROM TABLE(GetVendors);

create or replace package customer\_pkg is

type **customer\_row** is record(

cust\_name CUSTOMERS.CUST\_NAME%TYPE,

cust\_id CUSTOMERS.CUST\_ID%TYPE

);

type customers\_table\_info is table of **customer\_row**;

function **GetCustomers**(*count\_of\_orders* INT default 0)

return customers\_table\_info

**pipelined;**

end customer\_pkg;

create or replace package **body** customer\_pkg is

function **GetCustomers**(count\_of\_orders INT default 0)

return customers\_table\_info

**pipelined**

**IS**

**BEGIN**

**FOR customer\_iterator IN (**

**SELECT CUSTOMERS.CUST\_NAME, CUSTOMERS.CUST\_ID FROM CUSTOMERS**

**LEFT JOIN ORDERS**

**ON CUSTOMERS.CUST\_ID = ORDERS.CUST\_ID**

**GROUP BY CUSTOMERS.CUST\_NAME, CUSTOMERS.CUST\_ID**

**HAVING COUNT(ORDERS.CUST\_ID)=count\_of\_orders**

**)**

**LOOP**

**pipe row(customer\_iterator);**

**END LOOP;**

**END;**

end customer\_pkg;

select \* from TABLE(customer\_pkg.GetCustomers(1));

<https://docs.oracle.com/cd/B13789_01/appdev.101/b10802/t_anytyp.htm>

<https://docs.oracle.com/cd/B28359_01/appdev.111/b28425/pipe_paral_tbl_ref.htm>

create or replace type NColPipe as object

(

l\_parm varchar2(10), -- The parameter given to the table function

rows\_requested number, -- The parameter given to the table function

ret\_type anytype, -- The return type of the table function

rows\_returned number, -- The number of rows currently returned by the table function

static function ODCITableDescribe( rtype out anytype, p\_parm in varchar2, p\_rows\_req in number := 2 )

return number,

static function ODCITablePrepare( sctx out NColPipe, ti in sys.ODCITabFuncInfo, p\_parm in varchar2, p\_rows\_req in number := 2 )

return number,

static function ODCITableStart( sctx in out NColPipe, p\_parm in varchar2, p\_rows\_req in number := 2 )

return number,

member function ODCITableFetch( self in out NColPipe, nrows in number, outset out anydataset )

return number,

member function ODCITableClose( self in NColPipe )

return number,

static function show( p\_parm in varchar2, p\_rows\_req in number := 2 )

return anydataset pipelined using NColPipe

);

create or replace type body NColPipe as

static function ODCITableDescribe( rtype out anytype, p\_parm in varchar2, p\_rows\_req in number := 2 )

return number

is

atyp anytype;

begin

anytype.begincreate( dbms\_types.typecode\_object, atyp );

if p\_parm = 'one'

then

atyp.addattr( 'one'

, dbms\_types.typecode\_varchar2

, null

, null

, 10

, null

, null

);

elsif p\_parm = 'two'

then

atyp.addattr( 'one'

, dbms\_types.typecode\_varchar2

, null

, null

, 10

, null

, null

);

atyp.addattr( 'two'

, dbms\_types.typecode\_varchar2

, null

, null

, 10

, null

, null

);

else

atyp.addattr( p\_parm || '1'

, dbms\_types.typecode\_varchar2

, null

, null

, 10

, null

, null

);

atyp.addattr( p\_parm || '2'

, dbms\_types.typecode\_varchar2

, null

, null

, 10

, null

, null

);

atyp.addattr( p\_parm || '3'

, dbms\_types.typecode\_number

, 10

, 0

, null

, null

, null

);

end if;

atyp.endcreate;

anytype.begincreate( dbms\_types.typecode\_table, rtype );

rtype.SetInfo( null, null, null, null, null, atyp, dbms\_types.typecode\_object, 0 );

rtype.endcreate();

return odciconst.success;

exception

when others then

return odciconst.error;

end;

--

static function ODCITablePrepare( sctx out NColPipe, ti in sys.ODCITabFuncInfo, p\_parm in varchar2, p\_rows\_req in number := 2 )

return number

is

elem\_typ sys.anytype;

prec pls\_integer;

scale pls\_integer;

len pls\_integer;

csid pls\_integer;

csfrm pls\_integer;

tc pls\_integer;

aname varchar2(30);

begin

tc := ti.RetType.GetAttrElemInfo( 1, prec, scale, len, csid, csfrm, elem\_typ, aname );

sctx := NColPipe( p\_parm, p\_rows\_req, elem\_typ, 0 );

return odciconst.success;

end;

--

static function ODCITableStart( sctx in out NColPipe, p\_parm in varchar2, p\_rows\_req in number := 2 )

return number

is

begin

return odciconst.success;

end;

--

member function ODCITableFetch( self in out NColPipe, nrows in number, outset out anydataset )

return number

is

begin

anydataset.begincreate( dbms\_types.typecode\_object, self.ret\_type, outset );

for i in self.rows\_returned + 1 .. self.rows\_requested

loop

outset.addinstance;

outset.piecewise();

if self.l\_parm = 'one'

then

outset.setvarchar2( to\_char( i ) );

elsif self.l\_parm = 'two'

then

outset.setvarchar2( to\_char( i ) );

outset.setvarchar2( 'row: ' || to\_char( i ) );

else

outset.setvarchar2( 'row: ' || to\_char( i ) );

outset.setvarchar2( 'row: ' || to\_char( i ) );

outset.setnumber( i );

end if;

self.rows\_returned := self.rows\_returned + 1;

end loop;

outset.endcreate;

return odciconst.success;

end;

--

member function ODCITableClose( self in NColPipe )

return number

is

begin

return odciconst.success;

end;

end;

select \* from table( NColPipe.show( 'test', 3 ) );

**//TASK**

Вывести имя кастомера, количество заказов по условию количество заказов = перемен. Если условие не соблюдается, вывести кастомера и продукты, которые он не покупал.

СПОСОБЫ:

1. Создать три функции: 1) выгружает данные кастомер-ордер по условию; 2) выгружает кастомера и продукт; 3) объединяет эти две функции и формирует результат;
2. Одна функция, которая содержит два курсора: 1) пробегается по кастомерам, 2) пробегается по продуктам

***\*ТИПЫ НЕ ХРАНИТЬ В ПАКЕТАХ! НЕ ИСПОЛЬЗУЙ ПАКЕТЫ!!!!!!!!! ЗДЕСЬ!!!***

[***https://drive.google.com/file/d/0B8WDvzG3rLBfUFlMVDZXRldGWnM/view***](https://drive.google.com/file/d/0B8WDvzG3rLBfUFlMVDZXRldGWnM/view)

Синтаксис команды для создания триггера, следующий:

--- CREATE [OR REPLACE] TRIGGER имя\_триггера ---------------------  
--- BEFORE | AFTER активизирующее\_событие ON ссылка\_на\_таблицу ---  
--- FOR EACH ROW [WHEN условие\_срабатывания] ---------------------  
--- тело\_триггера ------------------------------------------------

Где:

* **имя\_триггера** - собственно имя вашего триггера.
* **активизирующее\_событие** - указывает момент активации триггера **BEFORE** до срабатывания оператора **DML**, **AFTER** после срабатывания оператора **DML**.
* **ссылка\_на\_таблицу** - собственно таблица, для которой создан триггер.
* **FOR EACH ROW** - если указано активируется от воздействия на строку если нет, то после любого оператора **DML**.
* **условие\_срабатывания** - если **TRUE** триггер срабатывает, если **FALSE** нет.
* **тело\_триггера** - собственно тело триггера.

create table user\_trigger

(

user\_trigger\_name varchar(50),

user\_trigger\_birthday date

);

CREATE OR REPLACE TRIGGER testTrg

AFTER INSERT OR DELETE OR UPDATE ON user\_trigger

DECLARE

BEGIN

DBMS\_OUTPUT.put\_line(‘testTrg fired’ );

END testTrg

|  |  |  |
| --- | --- | --- |
| **Активизирующий оператор** | **:OLD** | **:NEW** |
| INSERT | Не определена во всех полях содержится NULL значения | Значения, которые будут введены после выполнения оператора. |
| UPDATE | Исходные значения содержащиеся в строке перед обновлением данных | Новые значения которые будут введены после выполнения оператора |
| DELETE | Исходные значения содержащиеся в строке перед ее удалением | Не определена во всех полях содержится NULL значения |

create table user\_trigger

(

user\_trigger\_name varchar(50),

user\_trigger\_birthday date

);

create table user\_trigger\_log

(

info varchar(50)

);

CREATE OR REPLACE TRIGGER testTrg

AFTER INSERT OR DELETE OR UPDATE ON user\_trigger

DECLARE

BEGIN

insert into user\_trigger\_log (info ) VALUES (‘TRIGER FIRED’);

END testTrg

CREATE OR REPLACE TRIGGER trgNEW\_OLD\_Delete

BEFORE DELETE ON user\_trigger

FOR EACH ROW

DECLARE

user\_name user\_trigger.user\_trigger\_name %TYPE;

user\_birthday user\_trigger.user\_trigger\_birthday %TYPE;

BEGIN

user\_name :=:old.user\_trigger\_name;

DBMS\_OUTPUT.put\_line('user was deleted: '|| user\_name );

END trgNEW\_OLD\_Delete;

CREATE OR REPLACE TRIGGER trgNEW\_OLD\_Insert

BEFORE INSERT ON user\_trigger

FOR EACH ROW

DECLARE

user\_name user\_trigger.user\_trigger\_name %TYPE;

user\_birthday user\_trigger.user\_trigger\_birthday %TYPE;

BEGIN

user\_name :=:new.user\_trigger\_name;

if user\_name =’BOSS’ then

user\_name :=user\_name ||’!’

end if;

:new.user\_trigger\_name:=user\_name ;

DBMS\_OUTPUT.put\_line('user was deleted: '|| user\_name );

END trgNEW\_OLD\_Insert;

CREATE TABLE CUSTOMER\_REPORTS

( CUST\_ID char(10) NOT NULL ENABLE primary key,

CUST\_SUM NUMBER,

CUST\_DATE TIMESTAMP,

CUST\_STATUS VARCHAR(10)

);

CREATE OR REPLACE TRIGGER customer\_reports\_trg

BEFORE

INSERT OR

**UPDATE OF item\_price, quantity OR**

DELETE

ON ORDERITEMS

DECLARE

current\_cust\_id CUSTOMER\_REPORTS.CUST\_ID %TYPE;

operation\_type CUSTOMER\_REPORTS.CUST\_STATUS %TYPE;

count\_of\_row number;

BEGIN

SELECT COUNT(\*) into count\_of\_row from CUSTOMER\_REPORTS;

IF count\_of\_row >0 THEN

SELECT CUST\_ID into current\_cust\_id

FROM (SELECT CUST\_ID FROM CUSTOMER\_REPORTS ORDER BY CUST\_DATE DEsc )

WHERE rownum = 1;

ELSE

current\_cust\_id:=1;

END IF;

CASE

WHEN INSERTING THEN

operation\_type :='was insert';

WHEN UPDATING THEN

operation\_type :='was update';

WHEN DELETING THEN

operation\_type :='was delete';

END CASE;

INSERT INTO CUSTOMER\_REPORTS (CUST\_ID , CUST\_SUM , CUST\_DATE,CUST\_STATUS )

VALUES(TO\_CHAR(TO\_NUMBER(current\_cust\_id )+1), 0, sysdate, operation\_type );

END;

Create and drop sequences

CREATE SEQUENCE sequence\_name  
 MINVALUE value  
 MAXVALUE value  
 START WITH value  
 INCREMENT BY value  
 CACHE value;

MAXVALUE 999999999999999999999999999

CREATE SEQUENCE cust\_id\_seq  
 MINVALUE 1  
 MAXVALUE 1000000  
 START WITH 1  
 INCREMENT BY 1;

CREATE OR REPLACE TRIGGER customer\_reports\_trg

BEFORE

INSERT OR

**UPDATE OF item\_price, quantity OR**

DELETE

ON ORDERITEMS

DECLARE

operation\_type CUSTOMER\_REPORTS.CUST\_STATUS%TYPE;

BEGIN

CASE

WHEN INSERTING THEN

operation\_type :='was insert';

WHEN UPDATING THEN

operation\_type :='was update';

WHEN DELETING THEN

operation\_type :='was delete';

END CASE;

INSERT INTO CUSTOMER\_REPORTS (CUST\_ID , CUST\_SUM , CUST\_DATE,CUST\_STATUS )

VALUES(TO\_CHAR(cust\_id\_seq.NEXTVAL), 0, sysdate, operation\_type );

END;

CREATE VIEW view\_name AS  
 SELECT columns  
 FROM tables  
 [WHERE conditions];

create view as

select c.CUST\_NAME, o.ORDER\_NUM, SUM(QUANTITY \* ITEM\_PRICE) as OSUM

from ORDERS o

join ORDERITcust\_orders EMS oi on o.ORDER\_NUM = oi.ORDER\_NUM

join CUSTOMERS c on c.CUST\_ID = o.CUST\_ID

group by c.CUST\_ID, c.CUST\_NAME, o.ORDER\_NUM;

## **Instead of triggers**

CREATE OR REPLACE TRIGGER cust\_orders\_trg

INSTEAD OF INSERT ON cust\_orders

REFERENCING NEW as new\_order

FOR EACH ROW

DECLARE

current\_cust\_id ORDERS.CUST\_ID%TYPE;

cnt number;

BEGIN

select count(\*) into cnt from CUSTOMERS where CUST\_NAME = :new\_order.CUST\_NAME;

if cnt <> 1 then

ROLLBACK;

end if;

select CUST\_ID into current\_cust\_id

from CUSTOMERS

where CUST\_NAME = :new\_order.CUST\_NAME;

INSERT INTO ORDERS(ORDER\_NUM,ORDER\_DATE,CUST\_ID)

VALUES(:new\_order.ORDER\_NUM\*10,to\_date(sysdate),current\_cust\_id);

END;

insert into CUST\_ORDERS(CUST\_NAME, ORDER\_NUM) VALUES('Village Toys','20005');

Jobs

http://docs.oracle.com/cd/B19306\_01/appdev.102/b14258/d\_sched.htm#CHDBDCAI

CREATE TABLE CUSTOMER\_REPORTS

( CUST\_ID char(10) NOT NULL ENABLE primary key,

CUST\_SUM NUMBER,

CUST\_DATE TIMESTAMP,

CUST\_STATUS VARCHAR(10)

);

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

select o.CUST\_ID, o.ORDER\_NUM, sum(oi.ITEM\_PRICE \* oi.QUANTITY)

from ORDERS o

join ORDERITEMS oi on o.ORDER\_NUM = oi.ORDER\_NUM

where TO\_CHAR(ORDER\_DATE, 'mm') = TO\_CHAR(SYSDATE, 'mm') and TO\_CHAR(ORDER\_DATE, 'yyyy') = TO\_CHAR(SYSDATE, 'yyyy')

group by o.CUST\_ID, o.ORDER\_NUM;

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

alter table customer\_reports

drop constraint SYS\_C007267;

alter table customer\_reports

add constraint pk\_customer\_reports primary key(CUST\_ID, CUST\_DATE);

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

select o.CUST\_ID, sum(oi.ITEM\_PRICE \* oi.QUANTITY)

from ORDERS o

join ORDERITEMS oi on o.ORDER\_NUM = oi.ORDER\_NUM

where TO\_CHAR(ORDER\_DATE, 'mm') = TO\_CHAR(SYSDATE, 'mm') and TO\_CHAR(ORDER\_DATE, 'yyyy') = TO\_CHAR(SYSDATE, 'yyyy')

group by o.CUST\_ID;

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

create or replace procedure CUST\_SUM\_MONTH

as

begin

insert into CUSTOMER\_REPORTS(CUST\_ID, CUST\_SUM, CUST\_DATE, CUST\_STATUS)

select o.CUST\_ID, sum(oi.ITEM\_PRICE \* oi.QUANTITY), SYSDATE, null

from ORDERS o

join ORDERITEMS oi on o.ORDER\_NUM = oi.ORDER\_NUM

where EXTRACT(month FROM ORDER\_DATE) = EXTRACT(month FROM SYSDATE) and EXTRACT(year FROM ORDER\_DATE) = EXTRACT(year FROM SYSDATE)

group by o.CUST\_ID;

commit;

end;

-------

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

1. Create a program for the job:

BEGIN

DBMS\_SCHEDULER.CREATE\_PROGRAM(

program\_name => 'Insert\_Customer\_Orders',

program\_action => 'CUST\_SUM\_MONTH',

program\_type => 'STORED\_PROCEDURE'

);

COMMIT;

END;

2) Enable the program:

BEGIN

DBMS\_SCHEDULER.enable('Insert\_Customer\_Orders');

END;

3) Create a schedule for the job to run:

BEGIN

DBMS\_SCHEDULER.CREATE\_SCHEDULE (  
 schedule\_name => 'my\_schedule',  
 start\_date => SYSTIMESTAMP,  
 repeat\_interval => 'FREQ=MINUTELY; INTERVAL=1;',  
 end\_date => SYSTIMESTAMP + INTERVAL '30' day  
);

END;

4) Create your job with the program and schedule you defined:

BEGIN  
DBMS\_SCHEDULER.CREATE\_JOB (  
 job\_name => 'my\_user\_count\_job',  
 program\_name => 'Insert\_Customer\_Orders',  
 schedule\_name => 'my\_schedule');   
END;

5) Enable your job so it runs within the defined schedule

exec dbms\_scheduler.enable('my\_user\_count\_job')

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

select job\_name, status, run\_duration, cpu\_used  
from USER\_SCHEDULER\_JOB\_RUN\_DETAILS  
where job\_name = 'my\_user\_count\_job';

DBMS\_SCHEDULER.CREATE\_SCHEDULE (  
 schedule\_name IN VARCHAR2,  
 start\_date IN TIMESTAMP WITH TIMEZONE DEFAULT NULL,  
 repeat\_interval IN VARCHAR2,  
 end\_date IN TIMESTAMP WITH TIMEZONE DEFAULT NULL,  
 comments IN VARCHAR2 DEFAULT NULL);

create or replace trigger ActualSumQty

AFTER

UPDATE OF QUANTITY, ITEM\_PRICE OR

DELETE

ON ORDERITEMS FOR EACH ROW

DECLARE

CustID CUSTOMERS.CUST\_ID%TYPE;

DELTA ORDERITEMS.ITEM\_PRICE%TYPE;

BEGIN

CASE

WHEN UPDATING THEN

DELTA := :new.QUANTITY \* :new.ITEM\_PRICE - :old.QUANTITY \* :old.ITEM\_PRICE;

WHEN DELETING THEN

DELTA := - :old.QUANTITY \* :old.ITEM\_PRICE;

END CASE;

--

select CUST\_ID into CustID

from ORDERS o

WHERE o.ORDER\_NUM = :old.ORDER\_NUM;

--

update CUSTOMER\_REPORTS set CUST\_SUM = CUST\_SUM + DELTA, CUST\_STATUS = TO\_CHAR(DELTA) where CUST\_ID = CustID;

END;

Merger

MERGE INTO TABLE\_NAME   
USING table\_reference ON (condition)

WHEN MATCHED

THEN UPDATE SET column1 = value1 [, column2 = value2 ...]

WHEN NOT MATCHED   
THEN INSERT (column1 [, column2 ...]) VALUES (value1 [, value2 ...) ;

Jobs with parameters

CREATE OR REPLACE PROCEDURE ReducePrice(OrderNum number)

as

begin

update ORDERITEMS set ITEM\_PRICE = ITEM\_PRICE \* 0.9

where ORDER\_NUM = OrderNum;

end;

BEGIN  
DBMS\_SCHEDULER.**CREATE\_PROGRAM  
( program\_name => 'price\_update'**  
, program\_type => 'STORED\_PROCEDURE'

, program\_action => 'ReducePrice'  
, enabled => **false**

, number\_of\_arguments => 1  
);  
END;

BEGIN

DBMS\_SCHEDULER.DEFINE\_PROGRAM\_ARGUMENT

( program\_name => '**price\_update**'

, argument\_position => 1

, argument\_name => **'ReducePriceParam'**

, argument\_type => 'Number'

) ;

END;

BEGIN

DBMS\_SCHEDULER.CREATE\_SCHEDULE (  
 schedule\_name => 'my\_schedule\_with\_param',  
 start\_date => SYSTIMESTAMP,  
 repeat\_interval => 'FREQ=MINUTELY; INTERVAL=1;',  
 end\_date => SYSTIMESTAMP + INTERVAL '30' day  
);

END;

BEGIN  
DBMS\_SCHEDULER.CREATE\_JOB (  
 job\_name => 'my\_user\_price\_update\_job',  
 program\_name => **'price\_update'**,  
 schedule\_name => 'my\_schedule\_with\_param',

**enabled => FALSE**

);   
END;

BEGIN

DBMS\_SCHEDULER.SET\_JOB\_ANYDATA\_VALUE

( job\_name => 'my\_user\_price\_update\_job'

, argument\_name => **'ReducePriceParam'**

, argument\_value => ANYDATA.CONVERT**NUMBER** ( 20005 )

);

END;

EXECUTE DBMS\_SCHEDULER.ENABLE('price\_update');

EXECUTE DBMS\_SCHEDULER.**ENABLE**('my\_user\_price\_update\_job');

select \*

from USER\_SCHEDULER\_JOB\_RUN\_DETAILS;

-- Создаем хранимую процедуру

create or replace procedure ReducePriceByOrderProduct(OrderNum ORDERITEMS.ORDER\_NUM%TYPE, ProdId ORDERITEMS.PROD\_ID%TYPE)

as

begin

update ORDERITEMS set ITEM\_PRICE = ITEM\_PRICE \* 0.9 where ORDER\_NUM = OrderNum and PROD\_ID = ProdId;

end;

-- Создаем программу, которая ссылается на хранимую процедуру

BEGIN

DBMS\_SCHEDULER.CREATE\_PROGRAM

( program\_name => 'reduce\_price\_by\_order\_product'

, program\_type => 'STORED\_PROCEDURE' , program\_action => 'ReducePriceByOrderProduct'

, enabled => false

, number\_of\_arguments => 2

);

END;

BEGIN

DBMS\_SCHEDULER.DEFINE\_PROGRAM\_ARGUMENT

( program\_name => 'reduce\_price\_by\_order\_product'

, argument\_position => 1

, argument\_name => 'OrderNum'

, argument\_type => 'Number'

) ;

END;

BEGIN

DBMS\_SCHEDULER.DEFINE\_PROGRAM\_ARGUMENT

( program\_name => 'reduce\_price\_by\_order\_product'

, argument\_position => 2

, argument\_name => 'ProductId'

, argument\_type => 'Char'

) ;

END;

BEGIN

DBMS\_SCHEDULER.CREATE\_SCHEDULE (

schedule\_name => 'reduce\_price\_every\_minute',

start\_date => SYSTIMESTAMP,

repeat\_interval => 'FREQ=MINUTELY; INTERVAL=1;',

end\_date => SYSTIMESTAMP + INTERVAL '30' day

);

END;

BEGIN

DBMS\_SCHEDULER.CREATE\_JOB (

job\_name => 'reduce\_price\_job',

program\_name => 'reduce\_price\_by\_order\_product',

schedule\_name => 'reduce\_price\_every\_minute',

enabled => FALSE

);

END;

BEGIN

DBMS\_SCHEDULER.SET\_JOB\_ANYDATA\_VALUE

(

job\_name => 'reduce\_price\_job'

, argument\_name => 'ProductId'

, argument\_value => ANYDATA.CONVERTCHAR ( 'BR01' )

);

END;

BEGIN

DBMS\_SCHEDULER.SET\_JOB\_ANYDATA\_VALUE

(

job\_name => 'reduce\_price\_job'

, argument\_name => 'OrderNum'

, argument\_value => ANYDATA.CONVERTNUMBER ( 20005 )

);

END;

EXECUTE DBMS\_SCHEDULER.ENABLE('reduce\_price\_by\_order\_product');

EXECUTE DBMS\_SCHEDULER.ENABLE('reduce\_price\_job');

select \*

from USER\_SCHEDULER\_JOB\_RUN\_DETAILS;

BEGIN

DBMS\_SCHEDULER.CREATE\_JOB

( job\_name => 'simple\_job'

, job\_type => 'PLSQL\_BLOCK'

, job\_action => 'UPDATE ....;'

, enabled => TRUE

);

END;

BEGIN

DBMS\_SCHEDULER.CREATE\_JOB

( job\_name => 'simple\_job'

, job\_type => 'PLSQL\_BLOCK'

, job\_action => **q'[**UPDATE ORDERITEMS set ITEM\_PRICE = ITEM\_PRICE \* 1.5 where PROD\_ID = 'BR01' and ORDER\_NUM = 20005;**]'**

, schedule\_name => 'reduce\_price\_every\_minute'

, enabled => TRUE

);

END;

select \*

from USER\_SCHEDULER\_JOB\_RUN\_DETAILS where JOB\_NAME = 'SIMPLE\_JOB'

order by LOG\_DATE;

Oracle has a standard set of exceptions already named as follows:

|  |  |  |
| --- | --- | --- |
| **Oracle Exception Name** | **Oracle Error** | **Explanation** |
| DUP\_VAL\_ON\_INDEX | ORA-00001 | You tried to execute an INSERT or UPDATE statement that has created a duplicate value in a field restricted by a unique index. |
| TIMEOUT\_ON\_RESOURCE | ORA-00051 | You were waiting for a resource and you timed out. |
| TRANSACTION\_BACKED\_OUT | ORA-00061 | The remote portion of a transaction has rolled back. |
| INVALID\_CURSOR | ORA-01001 | You tried to reference a cursor that does not yet exist. This may have happened because you've executed a FETCH cursor or CLOSE cursor before OPENing the cursor. |
| NOT\_LOGGED\_ON | ORA-01012 | You tried to execute a call to Oracle before logging in. |
| LOGIN\_DENIED | ORA-01017 | You tried to log into Oracle with an invalid username/password combination. |
| NO\_DATA\_FOUND | ORA-01403 | You tried one of the following:   1. You executed a SELECT INTO statement and no rows were returned. 2. You referenced an uninitialized row in a table. 3. You read past the end of file with the UTL\_FILE package. |
| TOO\_MANY\_ROWS | ORA-01422 | You tried to execute a SELECT INTO statement and more than one row was returned. |
| ZERO\_DIVIDE | ORA-01476 | You tried to divide a number by zero. |
| INVALID\_NUMBER | ORA-01722 | You tried to execute a SQL statement that tried to convert a string to a number, but it was unsuccessful. |
| STORAGE\_ERROR | ORA-06500 | You ran out of memory or memory was corrupted. |
| PROGRAM\_ERROR | ORA-06501 | This is a generic "Contact Oracle support" message because an internal problem was encountered. |
| VALUE\_ERROR | ORA-06502 | You tried to perform an operation and there was a error on a conversion, truncation, or invalid constraining of numeric or character data. |
| CURSOR\_ALREADY\_OPEN | ORA-06511 | You tried to open a cursor that is already open. |

BEGIN  
 executable\_section  
  
EXCEPTION  
 WHEN exception\_name1 THEN  
 [statements]  
  
 WHEN exception\_name2 THEN  
 [statements]  
  
 WHEN exception\_name\_n THEN  
 [statements]  
  
 WHEN OTHERS THEN  
 [statements]  
  
END

create or replace PROCEDURE add\_order

IS

BEGIN

insert into ORDERS(ORDER\_NUM, ORDER\_DATE, CUST\_ID) VALUES(20005, SYSTIMESTAMP, 1000000001);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

raise\_application\_error (-20001,'You have tried to insert a duplicate');

END;

CREATE [OR REPLACE] PROCEDURE procedure\_name  
 [ (parameter [,parameter]) ]  
IS  
 [declaration\_section]  
  
 exception\_name EXCEPTION;  
  
BEGIN  
 executable\_section  
 RAISE exception\_name;  
  
EXCEPTION  
 WHEN exception\_name THEN  
 [statements]  
  
 WHEN OTHERS THEN  
 [statements]  
  
END [procedure\_name];

create or replace PROCEDURE add\_order\_with\_exeption

IS

order\_count Number;

order\_exeption EXCEPTION;

BEGIN

select COUNT(\*) into order\_count FROM ORDERS WHERE…

if ( order\_count >0) THEN

RAISE order\_exeption;

END IF;

insert into ORDERS(ORDER\_NUM, ORDER\_DATE, CUST\_ID) VALUES(20005, SYSTIMESTAMP, 1000000001);

EXCEPTION

WHEN order\_exeption THEN

raise\_application\_error (-20001,'You have tried to insert a duplicate');

END;

EXCEPTION  
 WHEN OTHERS THEN  
 raise\_application\_error(-20001,'An error was encountered - '||SQLCODE||' -ERROR- '||SQLERRM);  
  
END;

create or replace PROCEDURE add\_order2

IS

err\_code number;

err\_msg char(200);

BEGIN

insert into ORDERS(ORDER\_NUM, ORDER\_DATE, CUST\_ID) VALUES(20005, SYSTIMESTAMP, 1000000001);

EXCEPTION

WHEN OTHERS THEN

err\_code := SQLCODE;

err\_msg := SUBSTR(SQLERRM, 1, 200);

SYS.DBMS\_OUTPUT.PUT\_LINE('Message #1: ErrorCode' ||err\_code );

SYS.DBMS\_OUTPUT.PUT\_LINE('Message #2: ErrorMessage' || err\_msg );

END;

BEGIN

SYS.DBMS\_OUTPUT.ENABLE();

ADD\_ORDER2();

END;

create or replace PROCEDURE raiser

IS

BEGIN

raise\_application\_error (1,'error message');

END;

create or replace PROCEDURE client

IS

err\_code number;

err\_msg char(200);

BEGIN

raiser();

EXCEPTION

WHEN OTHERS THEN

err\_code := SQLCODE;

err\_msg := SUBSTR(SQLERRM, 1, 200);

SYS.DBMS\_OUTPUT.PUT\_LINE('Message #1: ErrorCode' ||err\_code || ' ' || err\_msg);

SYS.DBMS\_OUTPUT.PUT\_LINE('Message #2: ErrorCode' ||err\_code );

if (err\_code = -20000) then

SYS.DBMS\_OUTPUT.PUT\_LINE('Message #1: ErrorCode' ||err\_code );

elsif(err\_code = -20001) then

SYS.DBMS\_OUTPUT.PUT\_LINE('Message #2: ErrorCode' ||err\_code );

end if;

END;

Team Developing

DROP IF EXISTS

declare

counter int;

begin

select count(\*) into counter from **user\_tables** where table\_name = upper('table\_name');

if counter = 1 then

execute immediate 'drop table table\_name';

end if;

end;

create or replace procedure table\_droper(table\_name varchar2)

is

counter int;

begin

select count(\*) into counter from user\_tables where table\_name = upper(table\_name);

if counter = 1 then

execute immediate 'drop table '||table\_name;

end if;

end;

select \* from user\_tables;

DECLARE

TABLE\_NAME VARCHAR2(200);

BEGIN

TABLE\_NAME := 'Table1';

TABLE\_DROPER(

TABLE\_NAME => TABLE\_NAME

);

--rollback;

END;

select \* from user\_tables;

--------------Entity Definition

CREATE TABLE STUDENT

(

STUDENT\_EMAIL VARCHAR2(20) NOT NULL

, STUDENT\_NAME VARCHAR2(20) NOT NULL

);

--------------Relation Definition & Constratints

ALTER TABLE STUDENT

ADD CONSTRAINT student\_PK PRIMARY KEY (STUDENT\_EMAIL);

--------------Entity Definition

CREATE TABLE WORK

(

WORK\_DATE DATE NOT NULL

, STUDENT\_EMAIL\_FK VARCHAR2(20) NOT NULL

, WORK\_TEXT VARCHAR2(200) NOT NULL

, WORK\_MARK NUMBER

);

--------------Relation Definition & Constratints

ALTER TABLE WORK

ADD CONSTRAINT work\_PK PRIMARY KEY (WORK\_DATE,STUDENT\_EMAIL\_FK); -- primary key

ALTER TABLE WORK

ADD CONSTRAINT work\_FK FOREIGN KEY (STUDENT\_EMAIL\_FK) REFERENCES STUDENT (student\_email);

https://docs.google.com/document/d/1AKoJtSy4GXacWGM2cybrqAVkPLP4-5e-bJ8YcZhxrpY/edit#

student@K0601 MINGW64 ~

$ cd 'C:\Users\student.STEP\Desktop\MyDB'

student@K0601 MINGW64 ~/Desktop/MyDB

$ git config --global user.name "Igor"

student@K0601 MINGW64 ~/Desktop/MyDB

$ git config --global user.email tereshchenko.igor@gmail.com

student@K0601 MINGW64 ~/Desktop/MyDB

$ git config --global core.editor "'C:/Program Files/Notepad++/notepad++.exe' -m ultiInst -notabbar -nosession -noPlugin"

student@K0601 MINGW64 ~/Desktop/MyDB

$ git config --global alias.hist "log --pretty=format:'%h %ad | %s%d [%an]' --gr aph --date=short"

student@K0601 MINGW64 ~/Desktop/MyDB

$ git init

Initialized empty Git repository in C:/Users/student.STEP/Desktop/MyDB/.git/

student@K0601 MINGW64 ~/Desktop/MyDB (master)

$ git status

On branch master

Initial commit

Untracked files:

(use "git add <file>..." to include in what will be committed)

Entity Student.sql

Entity Work.sql

nothing added to commit but untracked files present (use "git add" to track)

student@K0601 MINGW64 ~/Desktop/MyDB (master)

$ git add 'Entity Student.sql'

student@K0601 MINGW64 ~/Desktop/MyDB (master)

$ git status

On branch master

Initial commit

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: Entity Student.sql

Untracked files:

(use "git add <file>..." to include in what will be committed)

Entity Work.sql

student@K0601 MINGW64 ~/Desktop/MyDB (master)

$ git add .

student@K0601 MINGW64 ~/Desktop/MyDB (master)

$ git status

On branch master

Initial commit

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: Entity Student.sql

new file: Entity Work.sql

student@K0601 MINGW64 ~/Desktop/MyDB (master)

$ git commit

[master (root-commit) 572ab24] commited Entities Student and Work

2 files changed, 36 insertions(+)

create mode 100644 Entity Student.sql

create mode 100644 Entity Work.sql

student@K0601 MINGW64 ~/Desktop/MyDB (master)

$ git hist

\* 572ab24 2017-04-21 | commited Entities Student and Work (HEAD -> master) [Igor ]

student@K0601 MINGW64 ~/Desktop/MyDB (master)

$ git tag -a version1

student@K0601 MINGW64 ~/Desktop/MyDB (master)

$ git hist

\* 572ab24 2017-04-21 | commited Entities Student and Work (HEAD -> master, tag: version1) [Igor]

student@K0601 MINGW64 ~/Desktop/MyDB (master)

$ git status

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: Entity Student.sql

no changes added to commit (use "git add" and/or "git commit -a")

student@K0601 MINGW64 ~/Desktop/MyDB (master)

$ git add .

student@K0601 MINGW64 ~/Desktop/MyDB (master)

$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

modified: Entity Student.sql

student@K0601 MINGW64 ~/Desktop/MyDB (master)

$ git commit

[master 686eacd] added comment to student

1 file changed, 1 insertion(+)

student@K0601 MINGW64 ~/Desktop/MyDB (master)

$ git hist

\* 686eacd 2017-04-21 | added comment to student (HEAD -> master) [Igor]

\* 572ab24 2017-04-21 | commited Entities Student and Work (tag: version1) [Igor]

student@K0601 MINGW64 ~/Desktop/MyDB (master)

$ git tag -a version1.1

student@K0601 MINGW64 ~/Desktop/MyDB (master)

$ git hist

\* 686eacd 2017-04-21 | added comment to student (HEAD -> master, tag: version1.1) [Igor]

\* 572ab24 2017-04-21 | commited Entities Student and Work (tag: version1) [Igor]

student@K0601 MINGW64 ~/Desktop/MyDB (master)

$ git checkout version1

Note: checking out 'version1'.

You are in 'detached HEAD' state. You can look around, make experimental

changes and commit them, and you can discard any commits you make in this

state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may

do so (now or later) by using -b with the checkout command again. Example:

git checkout -b <new-branch-name>

HEAD is now at 572ab24... commited Entities Student and Work

student@K0601 MINGW64 ~/Desktop/MyDB ((version1))

$ git checkout version1.1

Previous HEAD position was 572ab24... commited Entities Student and Work

HEAD is now at 686eacd... added comment to student

student@K0601 MINGW64 ~/Desktop/MyDB ((version1.1))

$ git remote

student@K0601 MINGW64 ~/Desktop/MyDB ((version1.1))

$ git remote add myrep https://github.com/itstepstudent/OracleIntegration.git

student@K0601 MINGW64 ~/Desktop/MyDB ((version1.1))

$ git remote

myrep

student@K0601 MINGW64 ~/Desktop/MyDB ((version1.1))

$ git pull myrep master --allow-unrelated-histories

From https://github.com/itstepstudent/OracleIntegration

\* branch master -> FETCH\_HEAD

Merge made by the 'recursive' strategy.

LICENSE | 674 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++1 file changed, 674 insertions(+)

create mode 100644 LICENSE

student@K0601 MINGW64 ~/Desktop/MyDB ((74ea80a...))

$ git hist

\* 74ea80a 2017-04-21 | Merge branch 'master' of https://github.com/itstepstudent/OracleIntegration into HEAD (HEA D) [Igor]

|\

| \* cb24c49 2017-04-21 | Initial commit (myrep/master) [itstepstudent]

\* 686eacd 2017-04-21 | added comment to student (tag: version1.1, master) [Igor]

\* 572ab24 2017-04-21 | commited Entities Student and Work (tag: version1) [Igor]

student@K0601 MINGW64 ~/Desktop/MyDB ((74ea80a...))

$