"IBM continues to build the past, Microsoft tries to hold us in present, and Oracle continues to drive us into the future..."

Rich Niemiec,
president of the International Oracle Users Group (IOUG)

PL/SQL

Dr. Leonid Khukhlovich

leonidkh@012.net.il

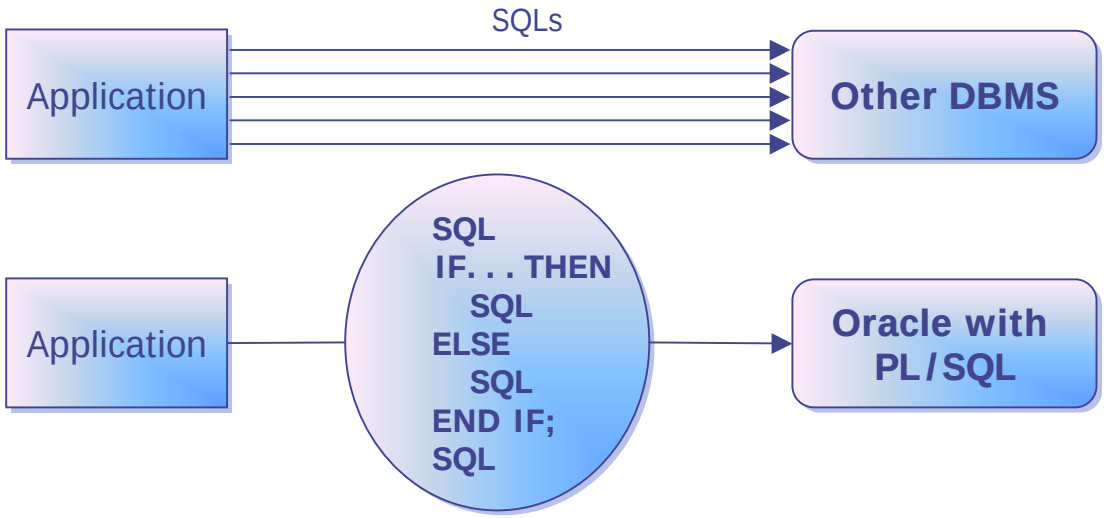
www.geocities.com/leokhg

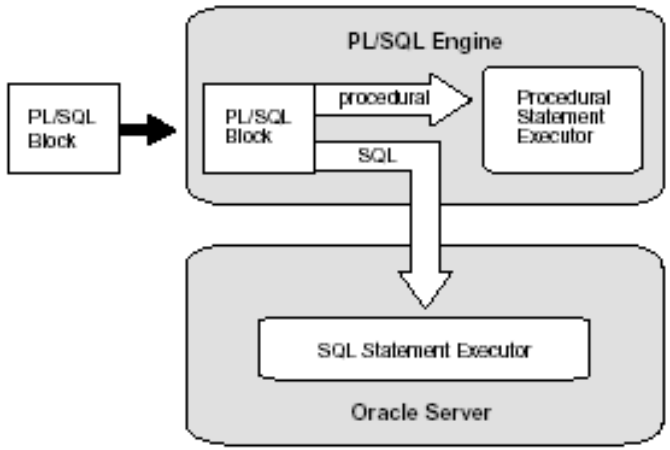
See also: **PL/SQL User's Guide and Reference**

Dr. Leonid Khukhlovich

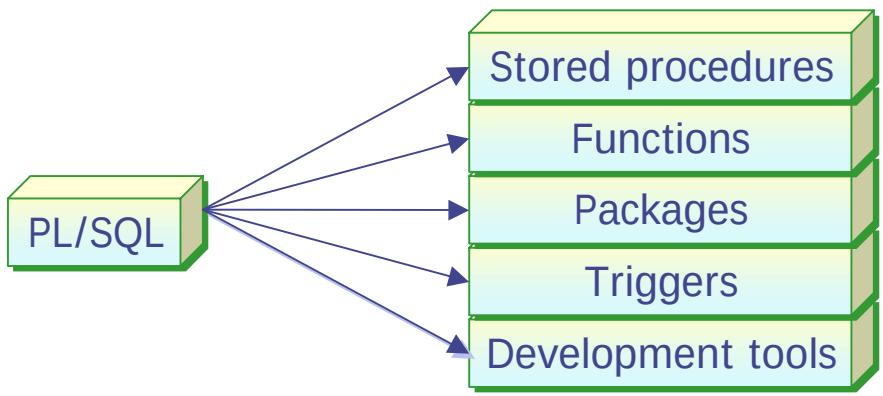
PL/SQL

- PL/SQL is an extension to SQL with design features of programming languages.
- Data manipulation and query statements of SQL are included within procedural units of code.





Procedural Language/SQL (PL/SQL) is Oracle Corporation's procedural language extension to SQL, the standard data access language for relational databases PL/SQL offers modern software engineering features such as data encapsulation. exception handling, information hiding, and object orientation, and so brings state-of-the-art programming to the Oracle Server and Toolset.



Dr. Leonid Khukhlovich

PL/SQL Block Structure

```
DECLARE
   o o o
BEGIN
   o o o
EXCEPTION
   o o o
END;
```

DECLARE
Variables, constants, cursors, user-defined exceptions referenced in the executable and declarative sections

- **BEGIN**
  - SQL statements
  - PL/SQL statements

EXCEPTION
Actions to perform when errors and abnormal conditions occur in the executable section

- **END;**

```
DECLARE
   v_variable    VARCHAR2( 5 );
BEGIN
   SELECT column_name
      INTO v_variable
   FROM table_name;
EXCEPTION
   WHEN exception_name THEN
      ...
END;
```

Block types

Anonymous

```
[DECLARE]
. . .
BEGIN
. . .
[EXCEPTION]
. . .
END;
```

Procedure

```
PROCEDURE name( arg_list )
IS
. . .
BEGIN
. . .
[EXCEPTION]
. . .
END;
```

Function

```
FUNCTION name( arg_list )
RETURN datatype
IS
. . .
BEGIN
. . .
[EXCEPTION]
. . .
END;
```

Dr. Leonid Khukhlovich

Declaring PL/SQL Variables

Syntax:

identifier [CONSTANT] ***datatype*** [NOT NULL] [:= | DEFAULT ***expr***];

Examples:

```
DECLARE
   v_hiredate DATE;
   v_deptno NUMBER (2) NOT NULL :=10;
   v_location VARCHAR2(13) := 'Atlanta';
   c_comm CONSTANT  NUMBER :=1400;
```

Base scalar datatypes

- **VARCHAR2(max_length)**
- **NUMBER[(p, s)]**
- **DATE**
- **CHAR[(length)]**
- **LONG**
- **BOOLEAN**
- **BINARY_INTEGER**
- **PLS_INTEGER**

In the syntax:

identifier	is the name of the variable (should not have the same name as name of table columns used in the block)
CONSTANT	constrains the variable so that its value cannot change (Constants must be initialized).
datatype	is a **scalar, composite, reference or LOB data type**
NOT NULL	constrains the variable so that it must contain a value (NOT NULL variables must be initialized).
expr	is any PL/SQL expression that can be a literal, another variable or an expression involving operators and functions

```
v_job          VARCHAR2(9);
v_count        BINARY_INTEGER := 0;
v_total_sal    NUMBER( 9, 2 ) := 0;
v_orderdate    DATE := SYSDATE + 7;
```

```
c_tax_rate     CONSTANT NUMBER( 3, 2 ) := 8.25;
v_valid        BOOLEAN NOT NULL := TRUE;
v_ename        emp.ename%TYPE;
v_balance      v_total_sal%TYPE;
```

Dr. Leonid Khukhlovich

PL/SQL Variables

Assignment: *indentifier* **:=** *expr* **;**

Where *expr ::= { constant | variable | function call | expression }* , *but not a database column !*

Composite datatypes

PL/SQL RECORD

1723644	TRUE	23-DEC-98	ATLANTA	

PL/SQL TABLE

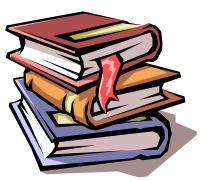
1	SMITH
2	JONES
3	NANCY
4	TIM

BINARY_INTEGER

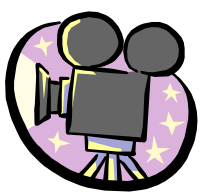
VARCHAR2

LOB datatypes

CLOB **BLOB** **BFILE**



NCLOB



The %TYPE prefix

- *tbl.column* **%TYPE**
- *previously_declared_var* **%TYPE**

```
DECLARE
  v_sum NUMBER( 9, 2 );
  v_id emp.empno%TYPE;
  v_total v.sum%TYPE;
BEGIN
  . . .
```

Dr. Leonid Khukhlovich

Comments

- **Single-line comments** — a := 0; -- *this is a comment*
- **Multi-line comments** — /* *All this text is comments* */

SQL Function in PL/SQL

- **Available in procedural statements:**
  - **Single-row string**
  - **Single-row number**
  - **Datatype conversion**
  - **Date**

  } Same as in SQL
- **Not available in procedural statements:**
  - **DECODE**
  - **Group function**

```
DECLARE
   v_hour    NUMBER(2);          -- for number of hours
   v_min     NUMBER(2);          -- for number of minutes
   v_SysD    DATE;               -- for system date
   v_res     VARCHAR2(16);       -- for result
BEGIN
   SELECT SYSDATE INTO v_SysD FROM DUAL;                  -- read system date
   v_hour    := TO_NUMBER( TO_CHAR( v_SysD, 'HH' ) ;      -- calculate number of hours
   v_min     := TO_NUMBER( TO_CHAR( v_SysD, 'MI' ) );     -- calculate number of minures
   v_res     := TRIM(TO_CHAR( v_hour, '00' )) || ':' ||
                TRIM(TO_CHAR( v_min, '00' )) ||
                ' ' || TO_CHAR( v_SysD, 'AM' );           -- compose result
   DBMS_OUTPUT.PUT_LINE( v_res );                         -- show result
END;
```

Dr. Leonid Khukhlovich

Nested Blocks

```
. . .
v_x NUMBER(5,2);
BEGIN
    . . .
    DECLARE
        v_x VARCHAR2( 20 );
        v_y DATE;
    BEGIN
        . . .
    END;
    . . .
END;
```

- Statements can be nested wherever an executable statement is allowed
- A nested block becomes a statement
- An exception section can contain nested blocks
- A block can look up to the enclosing block
- A block cannot look down to enclosed blocks

```
DECLARE
   v_sal    NUMBER(7,2) := 60000;
   v_comm   NUMBER(7,2) := V_SAL * 0.20;
   v_msg    VARCHAR2(255) := ' eligible for commision';
BEGIN . . .
   DECLARE
      v_sal    NUMBER(7,2) := 50000;
      v_comm   NUMBER(7,2) := 0;
      v_total  NUMBER(8,2) := v_sal + v_comm;
   BEGIN . . .
      v_msg := 'CLERK is not' || v_msg;
   END;
   v_msg := 'SALESMAN is' || v_msg;
END;
```

Dr. Leonid Khukhlovich

Logical flow control statement

```
IF condition THEN
        statements ;
[ ELSIF condition THEN
        statements ; ]
[ ELSE
        statements ; ]
END IF;
```

condition
{ **TRUE** | FALSE | NULL }

```
SET SERVEROUTPUT ON    -- SQL*Plus command
DECLARE
    v_cnt NUMBER(5);
    v_tot v_cnt%TYPE;
BEGIN
    SELECT COUNT( comm ), COUNT( * )
    INTO v_cnt, v_tot FROM emp;
    IF v_cnt = 0 THEN
        DBMS_OUTPUT.PUT_LINE( 'Nobody' );
    ELSIF v_cnt = 1 THEN
        DBMS_OUTPUT.PUT_LINE( 'Only one' );
    ELSIF v_cnt > 1 AND v_cnt <= v_tot / 2 THEN
        DBMS_OUTPUT.PUT_LINE( 'Less than half' );
    ELSIF v_cnt > v_tot / 2 AND v_cnt < v_tot THEN
        DBMS_OUTPUT.PUT_LINE( 'More than half' );
    ELSE
        DBMS_OUTPUT.PUT_LINE( 'Everybody' );
    END IF;
END;
```

Dr. Leonid Khukhlovich

Loop statements

Basic loop	**LOOP** *statement1* ; **. . .** **EXIT** [**WHEN** *condition*] **END LOOP** ;
FOR loop	**FOR** *counter* **IN** [**REVERSE**] *Lbound* **. .** *Ubound* **LOOP** *statement1* ; *statement2* ; **. . .** **END LOOP** ;
WHILE loop	**WHILE** *condition* **LOOP** *statement1* ; *statement2* ; **. . .** **END LOOP**;
Cursor **FOR** loop	?

```
DECLARE
   v_counter NUMBER(2) := 1;
BEGIN
   LOOP
      INSERT INTO item( order_id, item_id )
      VALUES( 622, v_counter );
      v_counter := v_counter + 1;
      EXIT WHEN v_counter > 10;
   END LOOP;
END;
-----------------------------
BEGIN
   FOR i IN 1..10 LOOP
      INSERT INTO item( order_id, item_id )
      VALUES( 622, i );
   END LOOP;
END;
-----------------------------
DECLARE
   v_counter NUMBER(2) := 1;
BEGIN
   WHILE v_counter <= 10 LOOP
      INSERT INTO item( order_id, item_id )
      VALUES ( 622, v_counter );
      v_counter := v_counter + 1;
   END LOOP;
END;
```

Dr. Leonid Khukhlovich

SQL statements in PL/SQL

```
SELECT select_list
INTO { variable_name [, variable_name ]...
       | record_name }
FROM table
WHERE condition
```

Query MUST return one and Only one row !

```
DECLARE
   v_deptno    NUMBER(2);
   v_loc       VARCHAR2(15);
BEGIN
   SELECT deptno, loc
   INTO v_deptno, v_loc
   FROM dept
   WHERE dname = 'SALES'
   . . .
END;
```

```
DECLARE
   v_sal_increase emp.sal%TYPE := 1000;
   v_new_empno emp.empno%TYPE;
   v_empty_dept dept.deptno%TYPE := 40;
BEGIN
   UPDATE EMP SET SAL = SAL + v_sal_increase
   WHERE JOB = 'ANALYST';

   SELECT NVL( MAX( EMPNO ) ) + 1 INTO v_new_empno
   FROM EMP;

   INSERT INTO EMP( EMPNO, ENAME, JOB, SAL, DEPTNO )
   VALUES( v_new_empno, 'ELLIS', 'CLERK', 1100, 20 );

   DELETE FROM DEPT
   WHERE DEPTNO = v_empty_dept;
END;
```

Dr. Leonid Khukhlovich

Sequences

```
CREATE SEQUENCE seq_name
[ INCREMENT BY integer ]
[ START WITH integer ]
[ MAXVALUE integer | NOMAXVALUE ]
[ MINVALUE integer | NOMINVALUE ]
[ CYCLE | NOCYCLE ]
[ CACHE | NOCACHE ];
```

```
ALTER SEQUENCE seq_name
[ INCREMENT BY integer ]
[ MAXVALUE integer | NOMAXVALUE ]
[ MINVALUE integer | NOMINVALUE ]
[ CYCLE | NOCYCLE ]
[ CACHE | NOCACHE ];
```

```
DROP SEQUENCE seq_name;
```

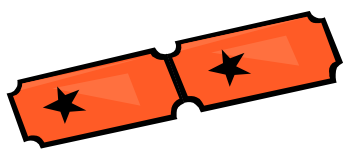
NEXTVAL

Returns the next available sequence value

CURRVAL

Obtainss the current sequence value

- Automatically generates unique numbers
- Is shared object
- Is typically used to create primary key value
- Replaces application code
- Speeds up the efficiency of accessing sequence values when cashed in memory

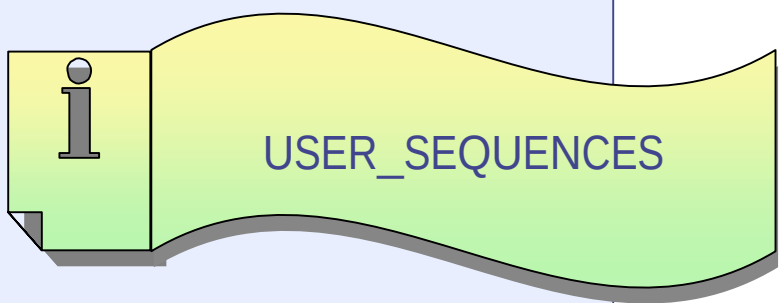


Dr. Leonid Khukhlovich

Sequences (cont.)

```
SQL> CREATE SEQUENCE DEPT_SEQ
  2 INCREMENT BY 10
  3 START WITH 50
  4 MINVALUE 10
  5 MAXVALUE 99;

Sequence created.
```

```
SQL> SELECT * FROM DEPT;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON

SQL> INSERT INTO DEPT VALUES( DEPT_SEQ.NEXTVAL, 'SOFTWARE', 'HUSTON' );

1 row created.

SQL> INSERT INTO DEPT VALUES( DEPT_SEQ.NEXTVAL, 'HARDWARE', 'CHICAGO' );

1 row created.

SQL> SELECT * FROM DEPT;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
        50 SOFTWARE       SACRAMENTO
        60 HARDWARE       CHICAGO
```



Dr. Leonid Khukhlovich

Triggers

A trigger defines an action the database should take when some of database-related events occurs

```
CREATE [ OR REPLACE ] TRIGGER trig_name
{ BEFORE | AFTER | INSTEAD OF }
{ DELETE | INSERT | UPDATE [ OF column [, column ] . . . ] }
[ OR { DELETE | INSERT | UPDATE [ OF column [, column ] . . . ] } ] . . .
ON { tab_name | view_name }
[ REFERENCING { OLD old | NEW new } ]
[ FOR EACH ROW ]
[ WHEN ( conditions ) ]
plsql_block
```

```
ALTER TRIGGER trig_name { ENABLE | DISABLE | COMPILE [ DEBUGG ] }
```

```
ALTER TABLE tab_name
{ ENABLE | DISABLE } ALL TRIGGERS
```

```
DROP TRIGGER trig_name
```

Dr. Leonid Khukhlovich

Using trigger for primary key generation

```
SQL> CREATE OR REPLACE TRIGGER trig_new_emp
  2  BEFORE INSERT ON DEPT
  3  FOR EACH ROW
  4  DECLARE
  5     v_dno NUMBER(2);
  6  BEGIN
  7     SELECT dept_seq.NEXTVAL
  8      INTO v_dno FROM dual;
  9     :NEW.DEPTNO := v_dno;
 10  END;
 11  /

Trigger created.

SQL> INSERT INTO DEPT( DNAME, LOC )
  2  VALUES( 'SOFTWARE', 'SACRAMENTO' );

1 row created.

SQL> INSERT INTO DEPT( DNAME, LOC )
  2  VALUES( 'HARDWARE', 'CHICAGO' );

1 row created.
```

```
SQL> SELECT * FROM DEPT;

    DEPTNO DNAME          LOC
---------- -------------- ----------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

```
SQL> SELECT * FROM DEPT;

    DEPTNO DNAME          LOC
---------- -------------- ----------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
        50 SOFTWARE       SACRAMENTO
        60 HARDWARE       CHICAGO
```

Dr. Leonid Khukhlovich

Using trigger for monitoring

```
SQL> CONNECT system/manager
Connected.
SQL> DROP TABLE secret1;

Table dropped.

SQL> CREATE TABLE secret1
  2  (
  3     vdate DATE,
  4     empno NUMBER(4),
  5     old_sal NUMBER(7,2),
  6     new_sal NUMBER(8,2),
  7     user_id NUMBER(5)
  8  );

Table created.

SQL> GRANT INSERT ON secret1
  2  TO scott;

Grant succeeded.
```

1

```
SQL> CONNECT scott/tiger
Connected.
SQL> CREATE OR REPLACE TRIGGER sal_secret1
  2  AFTER UPDATE OF sal
  3  ON emp
  4  FOR EACH ROW
  5  WHEN ( NEW.sal / OLD.sal > 1.1 )
  6  BEGIN
  7     INSERT INTO system.secret1
  8     VALUES( SYSDATE, :OLD.empno,
                :OLD.sal, :NEW.sal, UID );
  9  END;
 10  /

Trigger created.

SQL> UPDATE emp
  2  SET sal = sal + 200;

14 rows updated.

SQL> COMMIT;

Commit complete.
```

2

Dr. Leonid Khukhlovich

Using trigger for monitoring (cont.)

```
SQL> CONNECT system/manager
Connected.
SQL> SELECT * FROM secret1;

VDATE        EMPNO  OLD_SAL  NEW_SAL  USER_ID
--------- -------- -------- -------- --------
16-MAY-03     7369      800     1000       32
16-MAY-03     7499     1600     1800       32
16-MAY-03     7521     1250     1450       32
16-MAY-03     7654     1250     1450       32
16-MAY-03     7844     1500     1700       32
16-MAY-03     7876     1100     1300       32
16-MAY-03     7900      950     1150       32
16-MAY-03     7934     1300     1500       32

8 rows selected.
```

3

```
SQL> SELECT EMPNO, ENAME, SAL
  2  FROM SCOTT.EMP;

     EMPNO ENAME                SAL
---------- ---------- ----------
      7369 SMITH              1000
      7499 ALLEN              1800
      7521 WARD               1450
      7566 JONES              3175
      7654 MARTIN             1450
      7698 BLAKE              3050
      7782 CLARK              2650
      7788 SCOTT              3200
      7839 KING               5200
      7844 TURNER             1700
      7876 ADAMS              1300
      7900 JAMES              1150
      7902 FORD               3200
      7934 MILLER             1500

14 rows selected.
```

4

Cursors

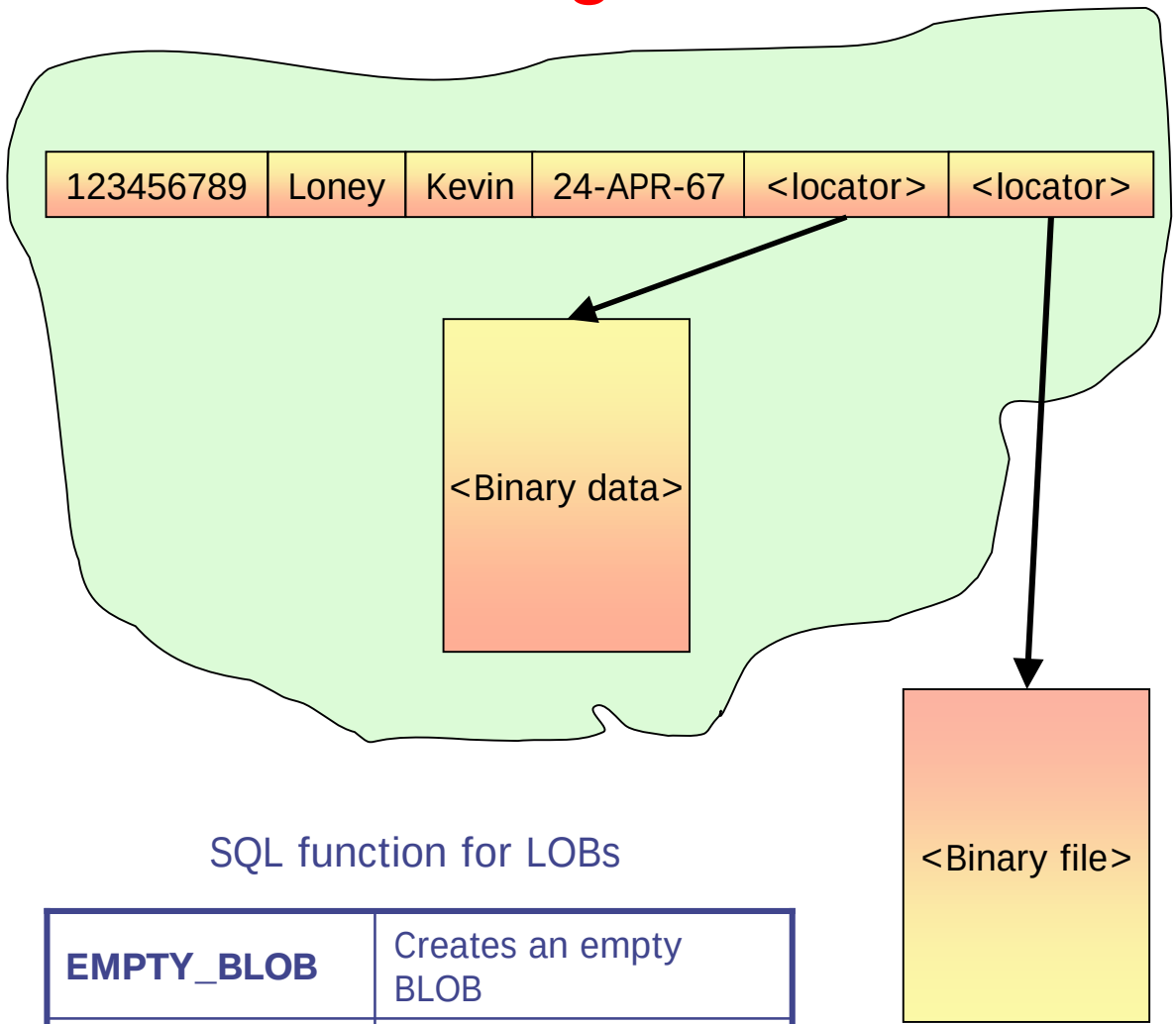
Dr. Leonid Khukhlovich

Exception handling

Dr. Leonid Khukhlovich

LOB fields handling

```
SQL> CREATE TABLE Person
  1  (
  2    ID NUMBER(9),
  3    LName VARCHAR2(15),
  4    FName VARCHAR2(12),
  5    BDate DATE,
  6    Photo BLOB,
  7    CV BFILE,
  8    PRIMARY KEY( ID )
  8  );

Table created.

SQL> INSERT INTO PERSON
  2  VALUES( 123456789,
  3          'Loney',
  4          'Kevin',
  5          '24-APR-67',
  6          EMPTY_BLOB(),
  7          NULL );

1 rows inserted.
```



SQL function for LOBs

EMPTY_BLOB	Creates an empty BLOB
EMPTY_CLOB	Create an empty CLOB
BFILENAME	Points to directory and file

Dr. Leonid Khukhlovich

BFILES and OS files

CREATE [**OR REPLACE**] **DIRECTORY** *directory*
AS '*pathname*'

Oracle alias of OS file

```
SQL> CREATE OR REPLACE DIRECTORY PersonDir
  1  AS
  2  'c:\LeoW\College\Oracle\LOBs';

Directory created

SQL> INSERT INTO Person
  2  VALUES( 125669, 'Loney',
             'Steven', '24-APR-67',
             EMPTY_BLOB(), NULL );

1 row created

SQL> UPDATE Person
  2  SET CV = BFILENAME( 'PersonDir', 'StevenCV.doc' )
  3  WHERE ID = 125669;

1 row updated
```

BFILENAME(p_d **IN VARCHAR2,**
p_f **IN VARCHAR2**)

DBMS_LOB Package

Procedures to modify BLOB and CLOB values

Function / Procedure	Description
APPEND()	Appends the LOB value to another LOB
COPY()	Copies all or part of a LOB to another LOB
ERASE()	Erases part of a LOB, starting at a specified offset
LOADFROMFILE()	Load BFILE data into an internal LOB
TRIM()	Trims the LOB value to the specified shorter length
WRITE()	Writes data to the LOB at a specified offset
WRITEAPPEND()	Writes data to the end of the LOB

DBMS_LOB Procedures to Read or Examine Internal and External LOB values

Function / Procedure	Description
COMPARE()	Compares the value of two LOBs
GETCHUNKSIZE()	Gets the chunk size used when reading and writing. This only works on internal LOBs and does not apply to external LOBs (BFILEs).
GETLENGTH()	Gets the length of the LOB value
INSTR()	Returns the matching position of the nth occurrence of the pattern in the LOB
READ()	Reads data from the LOB starting at the specified offset
SUBSTR()	Returns part of the LOB value starting at the specified offset

DBMS_LOB Procedures to Operate on Temporary LOBs

Function/Procedure	Description
CREATETEMPORARY()	Creates a temporary LOB
ISTEMPORARY()	Checks if a LOB locator refers to a temporary LOB
FREETEMPORARY()	Frees a temporary LOB

DBMS_LOB Read-Only Procedures for BFILEs

Function/Procedure	Description
FILECLOSE()	Closes the file. Use CLOSE() instead.
FILECLOSEALL()	Closes all previously opened files
FILEEXISTS()	Checks if the file exists on the server
FILEGETNAME()	Gets the directory alias and file name
FILEISOPEN()	Checks if the file was opened using the input BFILE locators. Use ISOPEN() instead.
FILEOPEN()	Opens a file. Use OPEN() instead.

DBMS_LOB Procedures to Open and Close Internal and External LOBs

Function / Procedure	Description
OPEN()	Opens a LOB
ISOPEN()	Sees if a LOB is open
CLOSE()	Closes a LOB