
ORACLE®

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.
The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

ORACLE

Can I do that? Introducing the Oracle Database 11*g* SQL and PL/SQL New Features

Stephen Jones
Practice Manager, Southeast Core Delivery, Oracle University

New Changes for SQL and PL/SQL

- Write SQL statements that include the new functions added to enhance regular expression support functionality
- Monitor dependency tracking and change notification
- New changes to locking that enable you to specify the maximum number of seconds the statement should wait to obtain a DML lock on the table
- Use the enhancements added to native dynamic SQL and to `DBMS_SQL`, which enable more interoperability between the two methodologies
- 11g Performance Enhancements

Regular Expression Enhancements in SQL and PL/SQL

- Features added:
  1. Access to the n-th subexpression in the `REGEXP_INSTR` and `REGEXP_SUBSTR` functions
  2. Return the number of times a pattern match is found in an input string using the new `REGEXP_COUNT` function

- Benefits:
  - Extends the current regular expression functionality based on customer feedback
  - Decreases the number of calls to the regular expression functions in order to get related information Third-level bullet

Using Subexpressions with Regular Expression Support

```
SELECT
  REGEXP_INSTR
① ('0123456789',       -- source char or search value
② '(123)(4(56)(78))', -- regular expression patterns
③ 1,                   -- position to start searching
④ 1,                   -- occurrence
⑤ 0,                   -- return option
⑥ 'i',                 -- match option (case insensitive)
⑦ 1)                   -- subexpression on which to search
"Position"
FROM dual;

  Position
----------
         2
```

- REGEXP_INSTR and REGEXP_SUBSTR now have an optional *subexpr* parameter that lets you target a particular substring of the regular expression being evaluated.

Why Access the n-th Subexpression?

- A more realistic use: DNA sequencing
- You may need to find a specific subpattern that identifies a protein needed for immunity in mouse DNA.

```
CREATE OR REPLACE FUNCTION get_instrsubexp_pos (p_subexp
    NUMBER)RETURN NUMBER
IS
  v_dna      CLOB;
  v_location NUMBER;
BEGIN
  v_dna := 'ccacctttccctccactcctcacgttctcacctgtaaagcgtccctccctcatccccatgccccttac
    cctgcagggtagagtaggctagaaaccagagagctccaagctccatctgtggagaggtgccatccttgggctgcagagagag
    gagaatttgccccaaagctgcctgcagagcttcaccacccttagtctcacaaagccttgagttcatagcatttctt
    gagttttcaccctgcccagcaggacactgcagcacccaaagggcttcccaggagtagggttgccctcaagaggctc
    ttgggtctgatggccacatcctggaattgttttcaagttgatggtcacagccctgaggcatgtaggggcgtgggga
    tgcgctctgctctgctctcctctcctgaacccctgaacccctctggctaccccagagcacttagagccag';
  v_location := REGEXP_INSTR(v_dna, '(gtc(tcac)(aaag))',
                1, 1, 0, 'i', p_subexp);
  RETURN (v_location);
END;
```

REGEXP_INSTR: Examples

```
SQL> EXECUTE dbms_output.put_line(get_instrsubexp_pos(1));
197

PL/SQL procedure successfully completed.

SQL> EXECUTE dbms_output.put_line(get_instrsubexp_pos(2));
200
PL/SQL procedure successfully completed.

SQL> EXECUTE dbms_output.put_line(get_instrsubexp_pos(3));
204

PL/SQL procedure successfully completed.
```

REGEXP_SUBSTR: Example

- REGEXP_SUBSTR searches for a regular expression pattern within a given string and returns the matched string.

```
SELECT
  REGEXP_SUBSTR
①('acgctgcactgca', -- source char or search value
②  'acg(.*)gca',    -- regular expression pattern
③  1,               -- position to start searching
④  1,               -- occurrence
⑤  'i',            -- match option (case insensitive)
⑥  1)              -- subexpression
   "Value"
FROM dual;

Value
------
ctgact
```

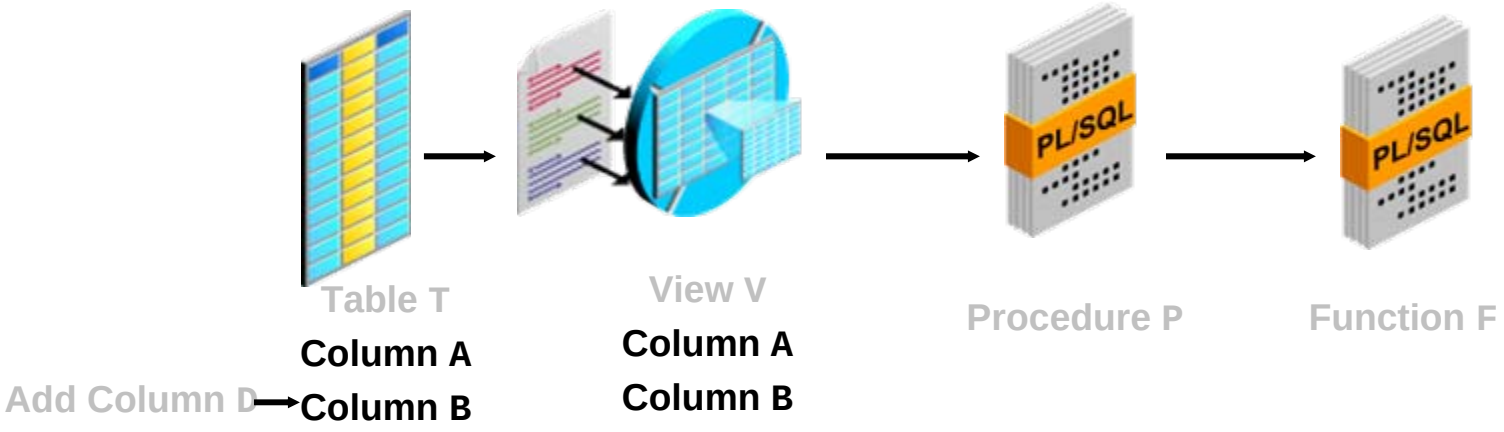
REGEXP_COUNT Function

- Returns the number of times a pattern appears in a string

```
CREATE OR REPLACE FUNCTION get_subexp_count
  (p_subexp VARCHAR2)RETURN NUMBER
IS
  v_dna      CLOB;
  v_count    NUMBER;
BEGIN
  v_dna :=   'ccacctttccctccactcctcacgttctcacctgtaaagcgtccctccctcatccccatgccccttaccctgcag
     ggtagagtaggctagaaaccagagagctccaagctccatctgtggagaggtgccatccttgggctgcagagagaggagaat
     ttgccccaaagctgcctgcagagcttcaccacccttagtctcacaaagccttgagttcatagcatttcttgagttttcacc
     ctgcccagcaggacactgcagcacccaaagggcttcccaggagtagggttgccctcaagaggctcttgggtctgatggcca
     catcctggaattgttttcaagttgatggtcacagccctgaggcatgtaggggcgtggggatgcgctctgctctgctctcct
     ctcctgaacccctgaaccctctggctaccccagagcacttagagccag';
  v_count := REGEXP_COUNT(v_dna, p_subexp);
  RETURN (v_count);
END;
```

More Precise Dependency Metadata

- Oracle Database 11*g* records additional, finer-grained dependency management.
- Earlier releases recorded dependency metadata.
- Prior to Oracle Database 11*g*, adding column `D` to table `T` invalidated the dependent objects.
- Starting in Oracle Database 11*g*, adding column `D` to table `T` does not impact view `V` and does not invalidate the dependent objects.



Fine-Grain Dependency Management

- In Oracle Database 11*g*, dependencies are tracked at the level of *element within unit.* Element-based dependency tracking covers the following:
  - Dependency of a single-table view on its base table
  - Dependency of a PL/SQL program unit (package specification, package body, or subprogram) on the following:
    - Other PL/SQL program units
    - Tables
    - Views

SQL Fine-Grain Dependency Management: Example

①

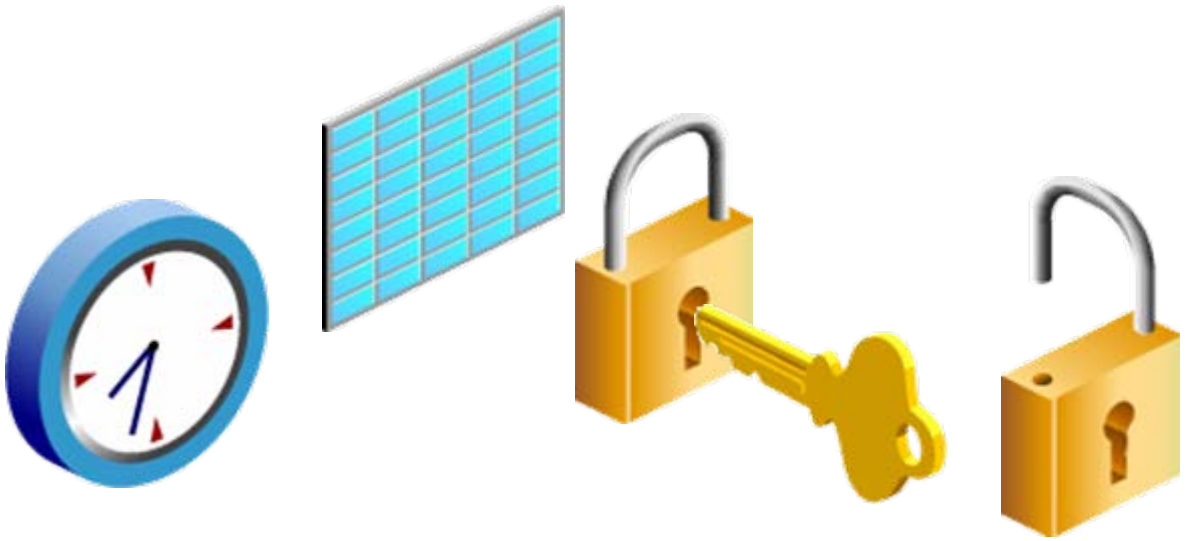
```
CREATE TABLE t (col_a NUMBER, col_b NUMBER, col_c NUMBER);
CREATE VIEW v AS SELECT col_a, col_b FROM T;

SELECT ud.name, ud.type, ud.referenced_name,
       ud.referenced_type, uo.status
FROM user_dependencies ud, user_objects uo
WHERE ud.name = uo.object_name AND ud.name = 'V';

NAME            TYPE       REFERENCED_NAME  REFERENCED_TYPE   STATUS
--------------- ---------- ---------------- ----------------- -------
V               VIEW       T                TABLE             VALID
```

②

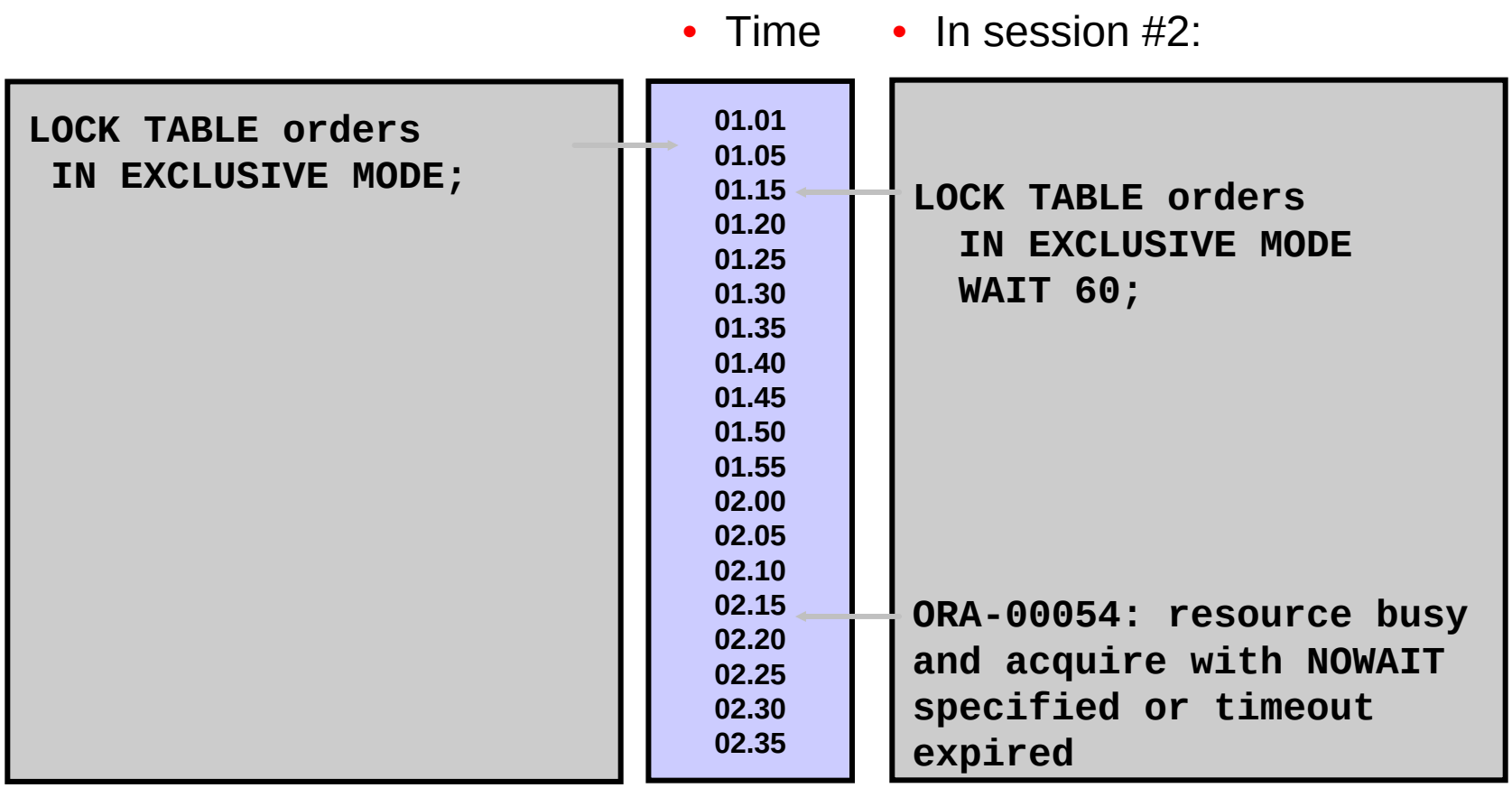
```
ALTER TABLE t ADD (col_d VARCHAR2(20));

SELECT ud.name, ud.type, ud.referenced_name,
       ud.referenced_type, uo.status
FROM user_dependencies ud, user_objects uo
WHERE ud.name = uo.object_name AND ud.name = 'V';

NAME            TYPE       REFERENCED_NAME  REFERENCED_TYPE   STATUS
--------------- ---------- ---------------- ----------------- -------
V               VIEW       T                TABLE             VALID
```

Using the LOCK TABLE Statement with the WAIT Option

- Use the WAIT option to identify the maximum number of seconds a statement should wait to obtain a DML lock on the table.
  - There is no limit on the number of seconds.
  - A message is returned indicating that the object is already locked.



Using the LOCK TABLE Statement with the WAIT Option: Example



Setting the DDL_LOCK_TIMEOUT Parameter

- Use the DDL_LOCK_TIMEOUT parameter to specify a DDL lock timeout.
  - The permissible range of values for DDL_LOCK_TIMEOUT is 0 through 1,000,000 (in seconds).
  - The default is 0.
- You can set DDL_LOCK_TIMEOUT at the:
  - System level
  - Session level (with an ALTER SESSION statement)

Executing Dynamic SQL in PL/SQL with the 11*g* Enhancements

Dynamic SQL Functional Completeness

- Currently have two flavors of dynamic SQL within PL/SQL: `DBMS_SQL` and native dynamic SQL
- For functional completeness, interoperability between native dynamic SQL and `DBMS_SQL` is supported:
  - SQL statements larger than 32 KB are allowed in native dynamic SQL.
  - `DBMS_SQL.PARSE()` is overloaded for CLOBs.
  - A `REF CURSOR` can be converted to a `DBMS_SQL` cursor and vice versa to support interoperability.
  - `DBMS_SQL` supports the full range of data types (including collections and object types).

Dynamic SQL Support for CLOBs

- Native dynamic SQL support:

```
CREATE OR REPLACE PROCEDURE gen_pl
(p_pgm CLOB)
IS
  dynamic_pl CLOB := p_pgm;
BEGIN
  EXECUTE IMMEDIATE dynamic_pl;
  -- next line is for learning purposes only
  DBMS_OUTPUT.PUT_LINE ('Just executed the following code: ' ||
    dynamic_pl);
END gen_pl;
```

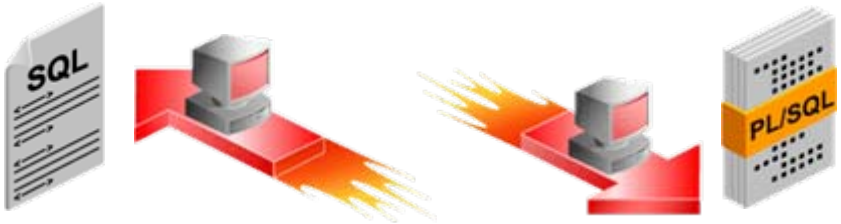
1

```
EXECUTE gen_pl('begin dbms_output.put_line –
  (''put any code here''); end;')

put any code here

Just executed the following code: begin dbms_output.put_line('put any
    code
here'); end;

PL/SQL procedure successfully completed.
```

2

Transforming a DBMS_SQL Cursor into a REF CURSOR

- To add interoperability between native dynamic SQL and DBMS_SQL, you can transform a DBMS_SQL cursor into a PL/SQL REF CURSOR and vice versa.



- Two new functions are added into the DBMS_SQL package to support this feature:
  - DBMS_SQL.TO_REFCURSOR
    (cursor_number IN INTEGER)
    RETURN SYS_REFCURSOR;
  - DBMS_SQL.TO_CURSOR_NUMBER
    (rc IN OUT SYS_REFCURSOR)
    RETURN INTEGER;

DBMS_SQL Support for Abstract Data Types (ADTs)

- Now allows:
  - Collections
    - Varrays
    - Nested tables
  - REFs
  - Opaque types

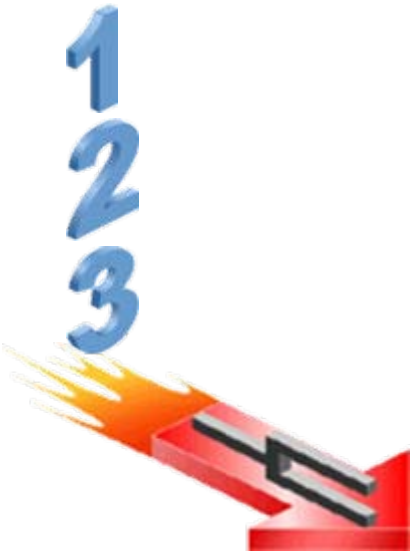
Implementing the Performance Improvements

Real Native Compilation

- The compiler translates PL/SQL source directly to the dynamic-link library (DLL) for the current hardware.
- The compiler does the linking and loading so that the file system directories are no longer needed.
- The PL/SQL native compilation works out of the box, without requiring a C compiler on a production box.
- The `PLSQL_CODE_TYPE` parameter is the on/off switch.
- And, real native compilation is faster than the C native compilation.

SIMPLE_INTEGER Data Type

- Definition:
  - Is a predefined subtype
  - Has the range –2147483648 .. 2147483648
  - Does not include a null value
  - Is allowed anywhere in PL/SQL where the PLS_INTEGER data type is allowed
- Benefits:
  - Eliminates the overhead of overflow checking
  - Is estimated to be 2–10 times faster when compared with the PLS_INTEGER type with native PL/SQL compilation

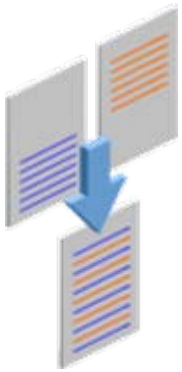


Intra Unit Inlining

- Definition:
  - Inlining is defined as the replacement of a call to subroutine with a copy of the body of the subroutine that is called.
  - The copied procedure generally runs faster than the original.
  - The PL/SQL compiler can automatically find the calls that should be inlined.
- Benefits:
  - Inlining can provide large performance gains when applied judiciously by a factor of 2–10 times.

Use of Inlining

- Influence implementing inlining via two methods:
  - Oracle parameter `PLSQL_OPTIMIZE_LEVEL`
  - `PRAGMA INLINE`
- Recommend that you:
  - Inline small programs
  - Inline programs that are frequently executed
- Use performance tools to identify hot spots suitable for inline applications:
  - `plstimer`



Inlining: Example

- Set the PLSQL_OPTIMIZE_LEVEL session-level parameter to a value of 2 or 3:

```
ALTER PROCEDURE small_pgm COMPILE
  PLSQL_OPTIMIZE_LEVEL = 3 REUSE SETTINGS;
```

  - Setting it to 2 means no automatic inlining is attempted.
  - Setting it to 3 means automatic inlining is attempted and no pragmas are necessary.
- Within a PL/SQL subroutine, use PRAGMA INLINE
  - NO means no inlining occurs regardless of the level and regardless of the YES pragmas.
  - YES means inline at level 2 of a particular call and increase the priority of inlining at level 3 for the call.

Inlining: Guidelines

- Pragmas apply only to calls in the next statement following the pragma.
- Programs that make use of smaller helper subroutines are good candidates for inlining.
- Only local subroutines can be inlined.
- You cannot inline an external subroutine.
- Cursor functions should not be inlined.
- Inlining can increase the size of a unit.
- Be careful about suggesting to inline functions that are deterministic.

SQL Query Result Cache

- Definition:
  - Cache the results of the current query or query fragment in memory and then use the cached results in future executions of the query or query fragments.
  - Cached results reside in the result cache memory portion of the SGA.
- Benefits:
  - Improved performance

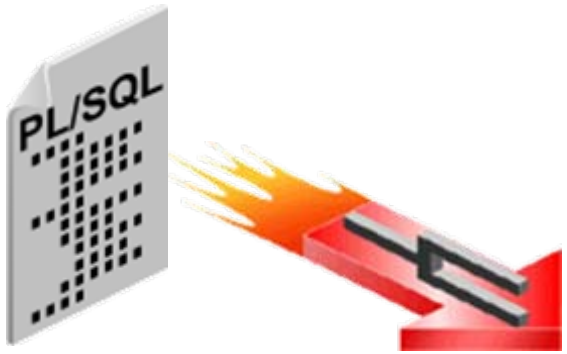
SQL Query Result Cache

- Scenario:
  - You need to find the greatest average value of credit limit grouped by state over the whole population.
  - The query results in a huge number of rows analyzed to yield a few or one row.
  - In your query, the data changes fairly slowly (say every hour) but the query is repeated fairly often (say every second).
- Solution:
  - Use the new optimizer hint `/*+ result_cache */` in your query:

```
SELECT /*+ result_cache */
  AVG(cust_credit_limit), cust_state_province
FROM sh.customers
GROUP BY cust_state_province;
```

PL/SQL Function Result Cache

- Definition:
  - Enables data that is stored in cache to be shared across sessions
  - Stores the function result cache in a shared global area (SGA), making it available to any session that runs your application
- Benefits:
  - Improved performance
  - Improved scalability



PL/SQL Function Result Cache

- Scenario:
  - You need a PL/SQL function that derives a complex metric.
  - The data that your function calculates changes slowly, but the function is frequently called.
- Solution:
  - Use the new `result_cache` clause in your function definition.

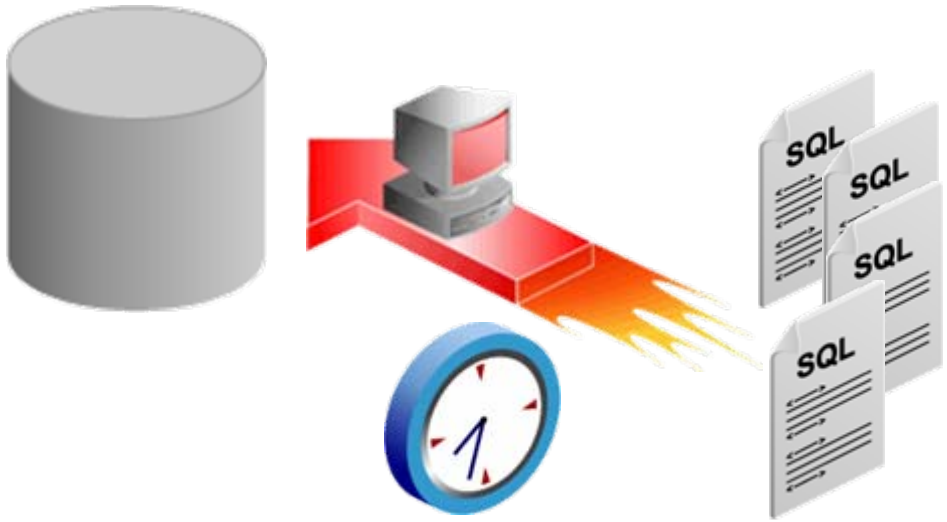
Enabling Result Caching

- Include the RESULT_CACHE option in the function definition.
- Optionally, include the RELIES_ON clause.

```
CREATE OR REPLACE FUNCTION productName
  (prod_id NUMBER, lang_id VARCHAR2)
  RETURN NVARCHAR2
  RESULT_CACHE RELIES_ON (product_descriptions)
IS
  result VARCHAR2(50);
BEGIN
  SELECT translated_name INTO result
    FROM product_descriptions
    WHERE product_id = prod_id AND language_id = lang_id;
  RETURN result;
END;
```

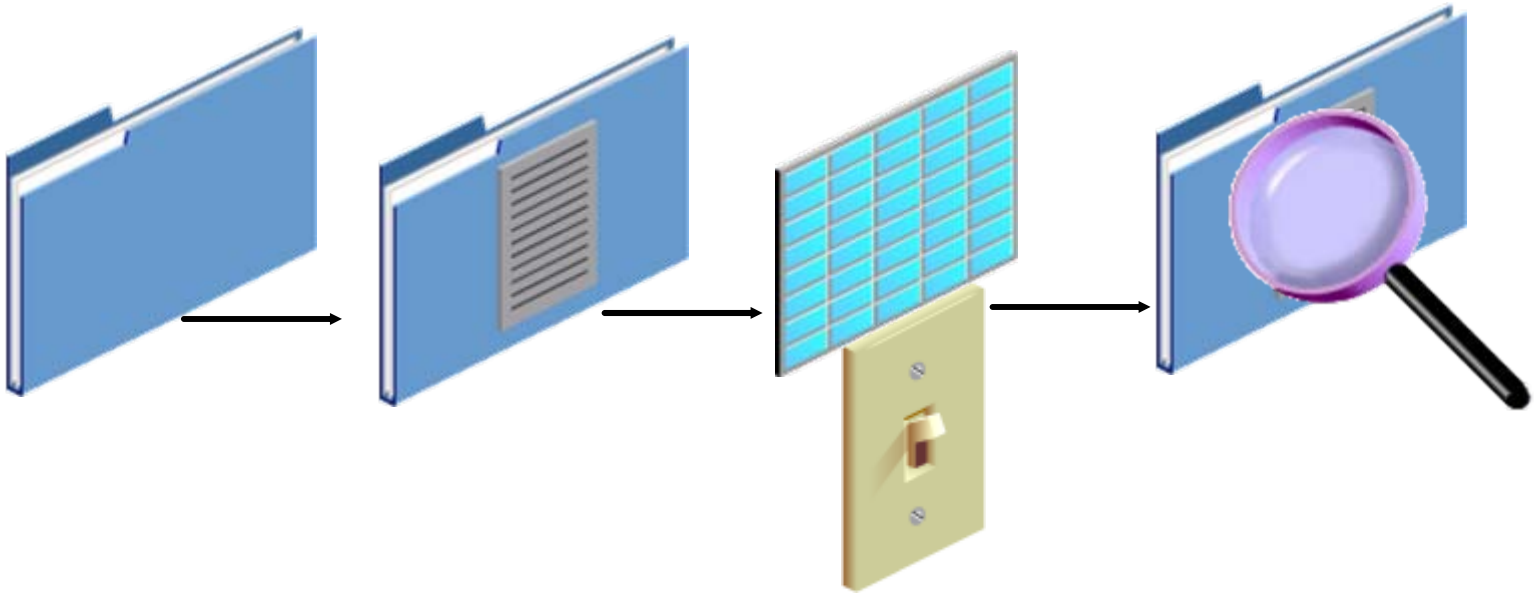
Flashback Data Archives

- Provide the ability to store and track all transactional changes to a record over its lifetime
- Save development resources because you no longer need to build this intelligence into your application
- Are useful for compliance with record stage policies and audit reports



Flashback Data Archive Process

1. Create the Flashback Data Archive.
2. Specify the default Flashback Data Archive.
3. Enable the Flashback Data Archive.
4. View Flashback Data Archive data.



Flashback Data Archive Scenario

- Using Flashback Data Archive to access historical data:

```
CONNECT sys/oracle@orcl AS sysdba
-- create the Flashback Data Archive
CREATE FLASHBACK ARCHIVE DEFAULT fla1
  TABLESPACE example QUOTA 10G RETENTION 5 YEAR;
```

1

```
-- Specify the default Flashback Data Archive
ALTER FLASHBACK ARCHIVE fla1 SET DEFAULT;
```

2

```
-- Enable Flashback Data Archive
ALTER TABLE oe1.inventories FLASHBACK ARCHIVE;
ALTER TABLE oe1.warehouses FLASHBACK ARCHIVE;
```

3

Flashback Data Archive Scenario

- Using Flashback Data Archive to access historical data:
  - Examine the data:

```
SELECT product_id, warehouse_id, quantity_on_hand
FROM oe1.inventories
WHERE product_id = 3108;
```
1

  - Change the data:

```
UPDATE oe1.inventories
SET quantity_on_hand = 300
WHERE product_id = 3108;
```
2

  - Examine the flashback data:

```
SELECT product_id, warehouse_id, quantity_on_hand
FROM oe1.inventories AS OF TIMESTAMP TO_TIMESTAMP
('2007-06-26 00:00:00', 'YYYY-MM-DD HH24:MI:SS')
WHERE product_id = 3108;
```
3

Flashback Data Archive DDL Restrictions

- Using any of the following DDL statements on a table enabled for Flashback Data Archive causes the error `ORA-55610: Invalid DDL statement on history-tracked table`
  - `ALTER TABLE` statement that does any of the following:
    - Drops, renames, or modifies a column
    - Performs partition or subpartition operations
    - Converts a `LONG` column to a LOB column
    - Includes an `UPGRADE TABLE` clause, with or without an `INCLUDING DATA` clause
  - `DROP TABLE` statement
  - `RENAME TABLE` statement
  - `TRUNCATE TABLE` statement

Oracle Database 11*g* SQL and PL/SQL Documentation

- *Oracle Database SQL Language Reference 11g, Release 1*
- *Oracle Database PL/SQL Language Reference 11g, Release 1*
- *Oracle Database PL/SQL Packages and Types Reference 11g, Release 1*
- *Oracle Database Large Objects Developer's Guide*
- *SQL*Plus User's Guide and Reference*
- *Oracle Database SQL Developer User's Guide, Release 1.2*

Additional Resources

- For additional information about the new Oracle 11*g* SQL and PL/SQL new features, refer to the following:
  - Oracle Database 11*g*: New Features eStudies
  - *Oracle by Example series (OBE): Oracle Database 11g*
    - http://www.oracle.com/technology/obe/11gr1_db/admin/11gr1db.html
  - What's New in PL/SQL in Oracle Database 11*g* on the Oracle Technology Network (OTN):
    - http://www.oracle.com/technology/tech/pl_sql/

For More Information

search.oracle.com

or

Education.oracle.com

ORACLE IS THE INFORMATION COMPANY


ORACLE®