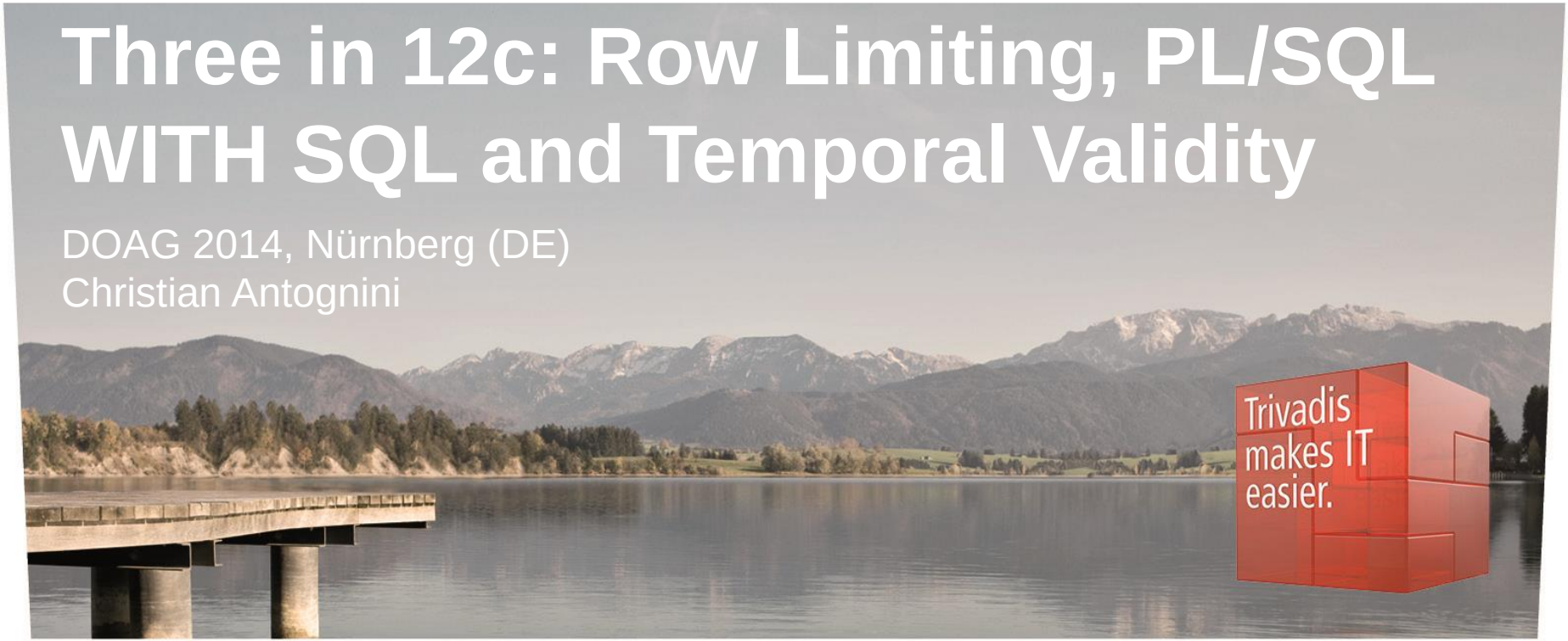
Three in 12c: Row Limiting, PL/SQL WITH SQL and Temporal Validity

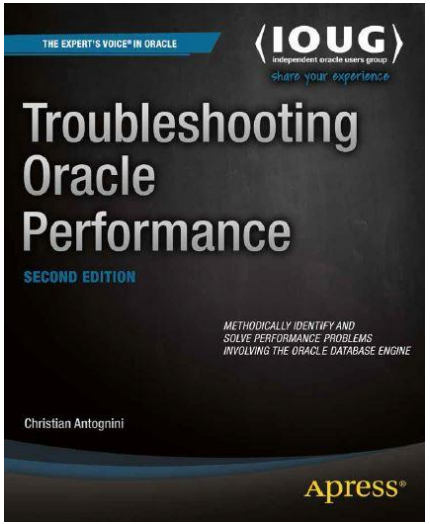
DOAG 2014, Nürnberg (DE)

Christian Antognini

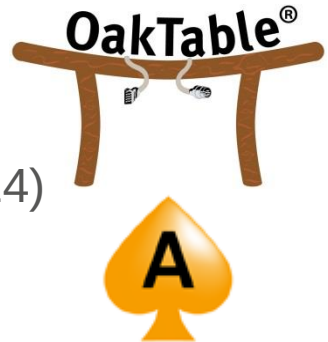


BASEL BERN BRUGG LAUSANNE ZUERICH DUESSELDORF FRANKFURT A.M. FREIBURG I.BR. HAMBURG MUNICH STUTTGART VIENNA

@ChrisAntognini



- Senior principal consultant, trainer and partner at Trivadis in Zurich (CH)
  - christian.antognini@trivadis.com
  - http://antognini.ch
- Focus: get the most out of Oracle Database
  - Logical and physical database design
  - Query optimizer
  - Application performance management
- Author of *Troubleshooting Oracle Performance* (Apress, 2008/2014)
- OakTable Network, Oracle ACE Director



AGENDA

1. Native Support for Top-N Queries
2. PL/SQL in SQL Statements
3. Temporal Validity

Native Support for Top-N Queries

ROW_LIMITING_CLAUSE

- The SELECT statement has been enriched by the ROW_LIMITING_CLAUSE
  - Based on SQL:2011
- Absolute top-N as well as percent top-N queries are possible
- Absolute OFFSET may be declared
- TIES handling is supported
- Data should be ordered to get a decent top-N result

Top-N Rows

ROW | ROWS can be used interchangeably

```
SELECT first_name, last_name, salary
  FROM hr.employees
 ORDER BY salary DESC
 FETCH FIRST 5 ROWS ONLY

FIRST_NAME           LAST_NAME                      SALARY
-------------------- ------------------------- ----------
Steven               King                            24000
Neena                Kochhar                         17000
Lex                  De Haan                         17000
John                 Russell                         14000
Karen                Partners                        13500
```

Absolute Offset

FIRST | NEXT can be used interchangeably

```
SELECT first_name, last_name
  FROM hr.employees
 ORDER BY salary DESC
OFFSET 5 ROWS FETCH NEXT 5 ROWS ONLY

FIRST_NAME           LAST_NAME                      SALARY
-------------------- ------------------------- ----------
Michael              Hartstein                       13000
Nancy                Greenberg                       12000
Shelley              Higgins                         12000
Alberto              Errazuriz                       12000
Lisa                 Ozer                            11500
```

Top Percent Rows

The number of rows returned is always a “ceiled” value in percent

```
SELECT first_name, last_name
  FROM hr.employees
 ORDER BY salary DESC
 FETCH FIRST 5 PERCENT ROWS ONLY

FIRST_NAME           LAST_NAME                      SALARY
-------------------- ------------------------- ----------
Steven               King                            24000
Neena                Kochhar                         17000
Lex                  De Haan                         17000
John                 Russell                         14000
Karen                Partners                        13500
Michael              Hartstein                       13000
```

No Percent Offset

```
SELECT first_name, last_name, salary
  FROM hr.employees
 ORDER BY salary DESC
 OFFSET 5 PERCENT ROWS FETCH NEXT 5 PERCENT ROWS ONLY

ORA-00905: missing keyword
```

- Also SQL:2011 doesn't have it

Ties Handling

- Use WITH TIES to specify how to handle equalities at the edge of the row set

```
SELECT first_name, last_name, salary
  FROM hr.employees
 ORDER BY salary DESC
 FETCH FIRST 1 PERCENT ROWS ONLY

FIRST_NAME           LAST_NAME                      SALARY
-------------------- ------------------------- ----------
Steven               King                            24000
Neena                Kochhar                         17000
```

Ties Handling

- Use WITH TIES to specify how to handle equalities at the edge of the row set

```
SELECT first_name, last_name, salary
  FROM hr.employees
 ORDER BY salary DESC
 FETCH FIRST 1 PERCENT ROWS WITH TIES

FIRST_NAME           LAST_NAME                      SALARY
-------------------- ------------------------- ----------
Steven               King                            24000
Neena                Kochhar                         17000
Lex                  De Haan                         17000
```

No Offset with Ties Handling

```
SELECT first_name, last_name, salary
  FROM employees
 ORDER BY salary DESC
OFFSET 2 ROWS WITH TIES FETCH NEXT 1 PERCENT ROWS WITH TIES

ORA-00933: SQL command not properly ended
```

- Also SQL:2011 doesn't have it

Performance

- Every implementation of a top-N query has to order the result set
  - Taking advantage of an index is also possible
- Either the WINDOW SORT or WINDOW SORT PUSHED RANK are used

```
-----------------------------------------------
| Id  | Operation                     | Name    |
-----------------------------------------------
|   0 | SELECT STATEMENT              |         |
|   1 |  VIEW                         |         |
|   2 |   WINDOW SORT PUSHED RANK     |         |
|   3 |    TABLE ACCESS FULL          | EMPLOYEES |
-----------------------------------------------
```

Assessment

- Top-N queries are easier using ROW_LIMITING_CLAUSE
  - No need to use inline views with RANK() / DENSE_RANK() / ROW_NUMBER() to determine the rank of a row
  - No need to use nested inline views using ORDER BY, ROWNUM
- Not being able to specify a percent OFFSET and an OFFSET with TIES might be a problem for paginating issues
  - But who does a paginating where the size of the returned dataset is unknown due to a per cent based specifications or an unknown number of ties?

PL/SQL in SQL Statements

New WITH Clause Feature

- It is possible to define PL/SQL functions and procedures within a SQL statement
  - They are only accessible inside this SQL statement

```
WITH FUNCTION round2five (in_amount NUMBER)
        RETURN NUMBER
     IS
     BEGIN
        RETURN ROUND(in_amount*20)/20;
     END;
SELECT last_name, first_name, salary
     , round2five(salary * 1.3131573) as new_sal_rounded
  FROM hr.employees
 WHERE ROWNUM <= 15
```

Options and Rules (1)

- More than one function/procedure may be defined
- Functions/procedures may access functions/procedures defined earlier within the same WITH clause
- Functions/procedures have to be defined prior to subquery factoring clause(s)
- Functions/procedures may be accessed by the subquery factoring clause(s)
- Functions/procedures specified in a WITH clause have priority over schema based functions/procedures sharing the same name
  - But not over SYS functions like USER/SYSDATE

Options and Rules (2)

- If the function/procedure declaration is not in a top level SELECT statement or is not part of a SELECT statement at all then the WITH_PLSQL hint has to be used
- Can be used in views
- AUTONOMOUS_TRANSACTION possible
- RESULT_CACHE is not supported in WITH clause functions

Example – Function with Exception Handler

```
WITH FUNCTION is_numeric (in_value IN VARCHAR2)
        RETURN NUMBER
     IS
        l_number number;
     BEGIN
        l_number := to_number(in_value);
        return 1;
     EXCEPTION
        WHEN value_error then return -1;
     END;
...
```

Example – Usage Within a View

```
CREATE OR REPLACE VIEW employee_view
AS
WITH FUNCTION get_manager_formatted (in_manager_id IN NUMBER)
       RETURN VARCHAR2
       DETERMINISTIC
     IS
...


SELECT *
  FROM employee_view
 WHERE rownum <= 15;
```

Example – AUTONOMOUS_TRANSACTION

```
CREATE OR REPLACE VIEW employee_view AS
WITH FUNCTION track_salary_access (in_rowid  IN ROWID
                                  ,in_salary IN NUMBER)
        RETURN NUMBER
     IS
        PRAGMA AUTONOMOUS_TRANSACTION;
     BEGIN
        INSERT INTO employee_log (row_id)
             VALUES (in_rowid);
        COMMIT;
        RETURN in_salary;
     END;
SELECT employee_id, first_name, last_name
     , track_salary_access(rowid, salary) salary
FROM hr.employees;
```

Performance

- Do not expect major differences
  - The code doesn't run in the SQL engine!
- WITH clause program units might be faster than schema-based program units
  - Only parameter passing was optimized
  - Equivalent performance can be achieved with schema-based program units by using the UDF pragma

Limitations / Assessment

- Cool stuff
- Whenever you use a function/procedure at more than one place, you should create a schema object to keep it maintainable
- SQL Developer 4.0.x does support this feature
- SQL*Plus prior to 12c does not support this feature

Temporal Validity

Temporal Validity versus Total Recall (FBDA)

Temporal Validity (Business Time)

Total Recall (System Time)

Temporal Validity versus Total Recall (FBDA)

Temporal Validity (Business Time)

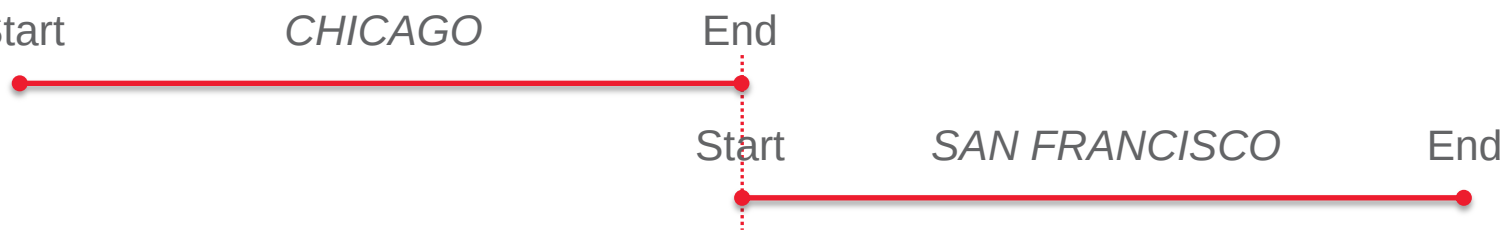
- Previous, current and future states
- Backdated changes are possible
- Future changes are possible
- Multiple valid time periods supported

Total Recall (System Time)

- Previous and current states
- Backdated changes are not possible[1)]
- Future changes are not possible[1)]
- Only system time supported

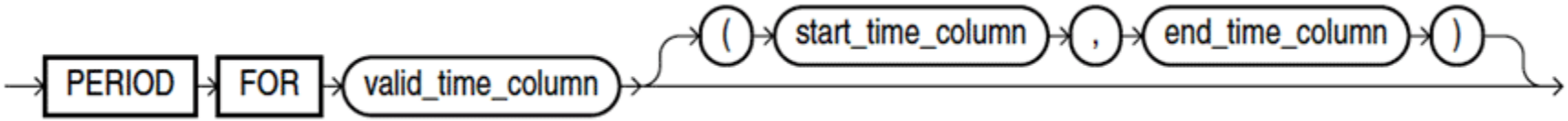
[1)] at least not as part of standard DML

Temporal Validity – Period, Semantics and Granularity



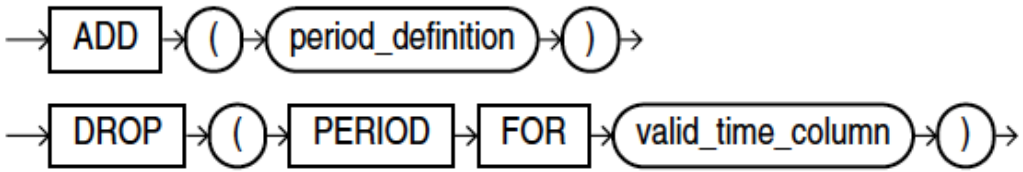
1981-02-20	2014-01-01	2015-04-01
1981-02-20 00:00:00	2014-01-01 00:00:00	2015-04-01 00:00:00
1981-02-20 00:00:00.000000	2014-01-01 00:00:00.000000	2015-04-01 00:00:00.000000

CREATE TABLE Enhancements



- *valid_time_column* is the name of temporal period
  - virtual and hidden
  - default prefix for *start_time_column* and *end_time_column*
- *start_time_column* and *end_time_column* (NULL = ∞)
  - hidden by default

ALTER TABLE Enhancements




```
ALTER TABLE dept ADD (vt_start DATE,
                      vt_end   DATE,
                      PERIOD FOR vt (vt_start, vt_end))
```

Visible Period Start and End Columns

```
SQL> SELECT * FROM dept;

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

Hidden, Virtual Column

```
SQL> SELECT column_name, data_type, data_length
          , hidden_column, virtual_column
       FROM user_tab_cols
      WHERE table_name = 'DEPT' AND column_name LIKE 'VT%';

COLUMN_NAME DATA_TYPE DATA_LENGTH HIDDEN_COLUMN VIRTUAL_COLUMN
----------- --------- ----------- ------------- --------------
VT          NUMBER             22 YES           YES
VT_START    DATE                7 NO            NO
VT_END      DATE                7 NO            NO
```

Constraint

```
SQL> SELECT table_name, constraint_type, search_condition
       FROM user_constraints
      WHERE table_name = 'DEPT';

TABLE_NAME CONSTRAINT_TYPE SEARCH_CONDITION
---------- --------------- --------------------------------
DEPT       C               (VT_START < VT_END) and (VT > 0)
```

AS OF PERIOD Query

```
SQL> SELECT *
  2    FROM dept AS OF PERIOD FOR vt DATE '2015-01-01'
  3   ORDER BY deptno;

    DEPTNO DNAME          LOC           VT_START   VT_END
---------- -------------- ------------- ---------- ----------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          SAN FRANCISCO 2014-01-01
        40 OPERATIONS     BOSTON
```

New Procedures in DBMS_FLASHBACK_ARCHIVE

- ENABLE_AT_VALID_TIME
  - LEVEL
    - ALL: no filter
    - CURRENT: filter records to display currently valid data
    - ASOF: filter records to display valid data as of <query_time>
  - QUERY_TIME
- DISABLE_ASOF_VALID_TIME
  - Clears an ASOF filter
  - Same as "ENABLE_AT_VALID_TIME('ALL')"

No DML Support – Do It Your Self…

```
SQL> UPDATE dept SET vt_end = DATE '2014-01-01' WHERE deptno = 30;

SQL> INSERT INTO dept (deptno, dname, loc, vt_start)
  2  VALUES (30, 'SALES', 'SAN FRANCISCO', DATE '2014-01-01');

SQL> SELECT *
       FROM dept
      WHERE deptno = 30
   ORDER BY vt_start NULLS FIRST;

 DEPTNO     DNAME          LOC            VT_START   VT_END
---------- -------------- ------------- ---------- ----------
        30 SALES          CHICAGO                  2014-01-01
        30 SALES          SAN FRANCISCO 2014-01-01
```

Limitations / Assessment

- Temporal validity is supported with Multitenant as of 12.1.0.2
  - In 12.1.0.1
    - DDL and DML works
    - Temporal flashback query filters are applied in a non-CDB instance only
- No support for temporal joins or temporal GROUP BY
- FLASHBACK_QUERY_CLAUSE
  - Temporal flashback query filters are not applied on views

Core Messages

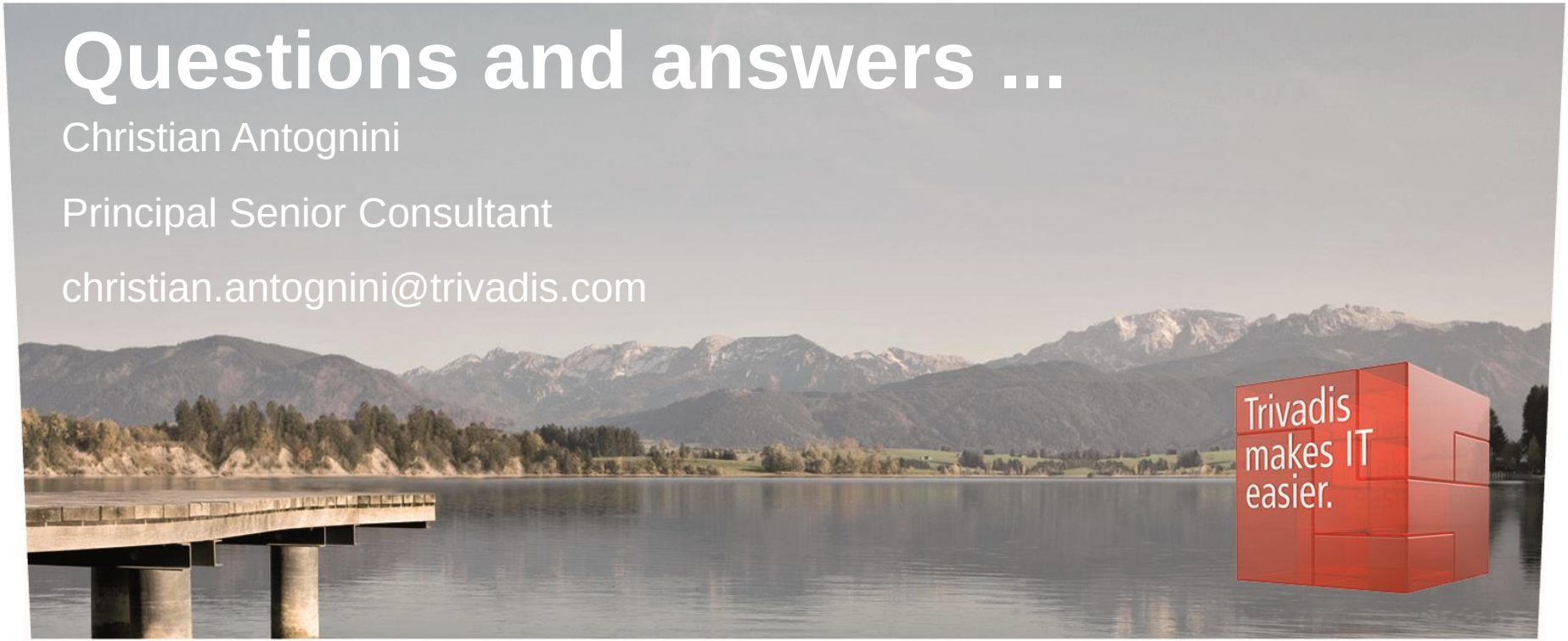
- Native Support for Top-N Queries
  - There is no good reason for not using it
- PL/SQL in SQL Statements
  - If correctly used, very good feature
- Temporal Validity
  - It's too soon!

Questions and answers ...

Christian Antognini

Principal Senior Consultant

christian.antognini@trivadis.com



BASEL BERN BRUGG LAUSANNE ZUERICH DUESSELDORF FRANKFURT A.M. FREIBURG I.BR. HAMBURG MUNICH STUTTGART VIENNA