Migrating Oracle queries to PostgreSQL

PGConf.EU 2012
Prague, Czech Republic
October 26th

Alexey Klyukin,
Command Prompt, Inc.

Photo by flickr member Nils Rinaldi
(http://www.flickr.com/people/nilsrinaldi/)

Why?

- Unsupported and outdated Oracle version
- PostgreSQL is mature
- Cost-effective
- Query conversion is easy

Why?

- Unsupported and outdated Oracle version
- PostgreSQL is mature
- Cost-effective
- Query conversion is easy, in theory :-)

- Oracle 8i (8.1.6) >>> PostgreSQL 9.1
- 500GB financial database
- Oracle-specific queries and data types
- No triggers or stored procedures
- Automated QA for the conversion

How?

- Data migration
- Query migration
- Reports
- Results comparison

Data migration

- CSV dump
- Ora2Pg
- Oracle Foreign Data Wrapper (FDW)
- Cross-database replication

Query conversion

- Oracle-style outer joins
- Pseudocolumns (i.e. ROWNUM)
- START WITH ... CONNECT BY
- Oracle-specific functions

Outer joins

- Oracle (+) syntax denotes the nullable side
- FULL OUTER JOINS are only possible via a hack in Oracle 8i and below
- Support for ANSI style JOINS introduced in Oracle 9i

Left outer joins

Oracle

SELECT * FROM
foo, bar WHERE
foo.baz = bar.baz (+)

PostgreSQL

SELECT * FROM
foo LEFT OUTER JOIN bar
ON (baz)

Right outer joins

Oracle	PostgreSQL
SELECT * FROM foo, bar, baz WHERE foo.id = bar.id (+) AND foo.id (+) = baz.id	SELECT * FROM foo LEFT OUTER JOIN bar ON (foo. id = bar.id) RIGHT OUTER JOIN baz ON (foo.id = baz.id)

Full outer joins

Oracle

SELECT * FROM
foo, bar WHERE
foo.id = bar.id (+)
UNION ALL
SELECT * FROM
foo, bar WHERE
foo.id (+) = bar.id AND
foo.id = NULL

PostgreSQL

SELECT * FROM
foo FULL OUTER JOIN bar
ON (foo.id = bar.id)

Pseudocolumns

- ROWID and ROWNUM
- CURRVAL and NEXTVAL
- LEVEL

Oracle ROWNUM

- Limiting the number of rows returned by a query
- Enumerating rows

ROWNUM vs LIMIT

Oracle

```
SELECT * FROM
foo
ORDER BY id
WHERE ROWNUM <= 10
```

PostgreSQL

```
SELECT * FROM
foo
ORDER BY id
LIMIT 10
```

ROWNUM vs LIMIT

Oracle

SELECT * FROM
foo
ORDER BY id
WHERE ROWNUM <= 10

ORDER BY is processed AFTER ROWNUM

PostgreSQL

SELECT * FROM
foo
ORDER BY id
LIMIT 10

ROWNUM vs LIMIT

Oracle

```
SELECT *
(SELECT * FROM
foo
ORDER BY id)
WHERE ROWNUM <= 10
```

PostgreSQL

```
SELECT * FROM
foo
ORDER BY id
LIMIT 10
```

Enumerating rows

- In Oracle — ROWNUM:

  SELECT ROWNUM, id FROM foo;

  UPDATE foo SET bar = bar ||'# '|| ROWNUM
- In PostgreSQL — window functions

Enumerating rows

- Window functions - PostgreSQL 8.4 and above (SQL:2003 standard compliant)
- Calculation over a set of rows
- Like aggregates, but without grouping the output into a single row
- Supported in Oracle 9i and above

Enumerating rows

Oracle	PostgreSQL
SELECT ROWNUM, foo FROM bar ORDER BY id	SELECT row_number() OVER (ORDER BY id) as rownum, foo FROM bar ORDER BY id

Row physical address

- Oracle — ROWID
  OOOOOO.FFF.BBBBBB.RRR
  (OBJECT.FILE.BLOCK.ROW)
- PostgreSQL — CTID (block no, tuple index)
- Identify rows uniquely
- Not persistent, not usable as a key

ROWID vs CTID

Oracle

DELETE FROM duplicates
WHERE ROWID =
(SELECT min(ROWID)
from duplicates)

PostgreSQL

DELETE FROM duplicates
WHERE ctid = (SELECT
min(ctid) FROM duplicates)

CONNECT BY

- Traverses hierarchical data
- Supports advanced features like cycle detections
- Oracle-specific

CONNECT BY

CREATE TABLE staff

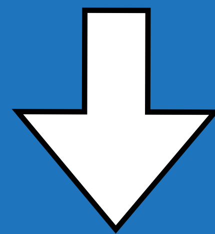
(id NUMBER PRIMARY KEY, name VARCHAR2(100), manager_id NUMBER)

SELECT name FROM staff START WITH name = 'John' CONNECT BY manager_id = PRIOR id

Gets all direct or indirect subordinates of John

"CONNECT BY" EXAMPLE: STEP 1

SELECT name FROM staff
START WITH name='John'
CONNECT BY
MANAGER_ID = PRIOR ID

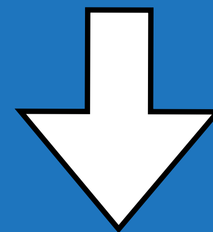


ID	1	2	3	4	5
NAME	John	Paul	Anna	Peter	Steve
MANAGER_ID		1	1	2	4

RESULT: ('John')

"CONNECT BY" EXAMPLE: STEP 2

SELECT name FROM staff
START WITH name='John'
CONNECT BY
MANAGER_ID = PRIOR ID

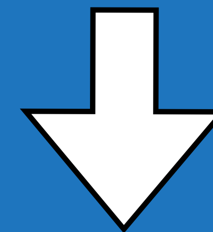


ID	1	2	3	4	5
NAME	John	Paul	Anna	Peter	Steve
MANAGER_ID		1	1	2	4

RESULT: ('John', 'Paul')

"CONNECT BY" EXAMPLE: STEP 3

SELECT name FROM staff
START WITH name='John'
CONNECT BY
MANAGER_ID = PRIOR ID

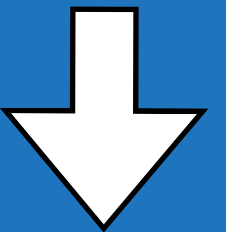


ID	1	2	3	4	5
NAME	John	Paul	Anna	Peter	Steve
MANAGER_ID		1	1	2	4

RESULT: ('John', 'Paul', 'Peter')

"CONNECT BY" EXAMPLE: STEP 4

SELECT name FROM staff
START WITH name='John'
CONNECT BY
MANAGER_ID = PRIOR ID

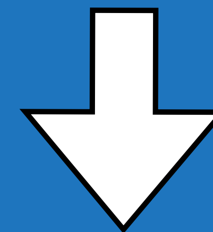


ID	1	2	3	4	5
NAME	John	Paul	Anna	Peter	Steve
MANAGER_ID		1	1	2	4

RESULT: ('John', 'Paul', 'Peter', 'Steve')

"CONNECT BY" EXAMPLE: STEP 5

```
SELECT name FROM staff
START WITH name='John'
CONNECT BY
MANAGER_ID = PRIOR ID
```



ID	1	2	3	4	5
NAME	John	Paul	Anna	Peter	Steve
MANAGER_ID		1	1	2	4

RESULT: ('John', 'Paul', 'Peter', 'Steve', 'Anna')

"CONNECT BY" EXAMPLE: FINISH

```
SELECT name FROM staff
START WITH name='John'
CONNECT BY
MANAGER_ID = PRIOR ID
```

ID	1	2	3	4	5
NAME	John	Paul	Anna	Peter	Steve
MANAGER_ID		1	1	2	4

RESULT: ('John', 'Paul', 'Peter', 'Steve', 'Anna')

Recursive Common Table Expressions (CTEs)

- AKA ‘WITH RECURSIVE’ queries
- Supported since PostgreSQL 8.4
- SQL compliant way of dealing with hierarchical data
- Very powerful

WITH RECURSIVE

CREATE TABLE staff (id INTEGER PRIMARY KEY, name TEXT, manager_id INTEGER)

WITH RECURSIVE st (id, name, manager_id) AS (SELECT id, name, manager_id FROM staff where name = 'John'

UNION ALL

SELECT id, name, manager_id FROM staff cur, st prev WHERE cur.manager_id = prev.id)

SELECT * FROM st

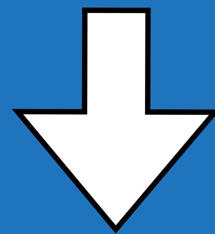
Recursive CTE EXAMPLE: STEP 1

WITH RECURSIVE st (id, name, manager_id) AS (SELECT id, name, manager_id FROM staff where name = 'John'

UNION ALL

SELECT id, name, manager_id FROM staff cur, st prev WHERE cur.manager_id = prev.id)

SELECT * FROM st



ID	1	2	3	4	5
NAME	John	Paul	Anna	Peter	Steve
MANAGER_ID		1	1	2	4

RESULT: ('John')

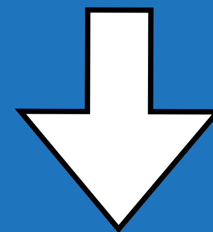
Recursive CTE EXAMPLE: STEP 2

WITH RECURSIVE st (id, name, manager_id) AS (SELECT id, name, manager_id FROM staff where name = 'John'

UNION ALL

SELECT id, name, manager_id FROM staff cur, st prev WHERE cur.manager_id = prev.id)

SELECT * FROM st



ID	1	2	3	4	5
NAME	John	Paul	Anna	Peter	Steve
MANAGER_ID		1	1	2	4

RESULT: ('John', 'Paul')

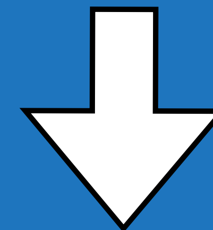
Recursive CTE EXAMPLE: STEP 3

WITH RECURSIVE st (id, name, manager_id) AS (SELECT id, name, manager_id FROM staff where name = 'John'

UNION ALL

SELECT id, name, manager_id FROM staff cur, st prev WHERE cur.manager_id = prev.id)

SELECT * FROM st



ID	1	2	3	4	5
NAME	John	Paul	Anna	Peter	Steve
MANAGER_ID		1	1	2	4

RESULT: ('John', 'Paul', 'Anna')

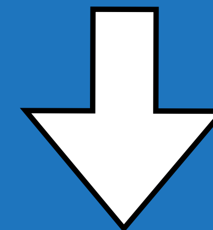
Recursive CTE EXAMPLE: STEP 4

WITH RECURSIVE st (id, name, manager_id) AS (SELECT id, name, manager_id FROM staff where name = 'John'

UNION ALL

SELECT id, name, manager_id FROM staff cur, st prev WHERE cur.manager_id = prev.id)

SELECT * FROM st



ID	1	2	3	4	5
NAME	John	Paul	Anna	Peter	Steve
MANAGER_ID		1	1	2	4

RESULT: ('John', 'Paul', 'Anna', 'Peter')

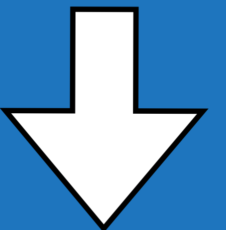
Recursive CTE EXAMPLE: STEP 5

WITH RECURSIVE st (id, name, manager_id) AS (SELECT id, name, manager_id FROM staff where name = 'John'

UNION ALL

SELECT id, name, manager_id FROM staff cur, st prev WHERE cur.manager_id = prev.id)

SELECT * FROM st



ID	1	2	3	4	5
NAME	John	Paul	Anna	Peter	Steve
MANAGER_ID		1	1	2	4

RESULT: ('John', 'Paul', 'Anna', 'Peter', 'Steve')

Recursive CTE example: FINISH

WITH RECURSIVE st (id, name, manager_id) AS (SELECT id, name, manager_id FROM staff where name = ‘John’

UNION ALL

SELECT id, name, manager_id FROM staff cur, st prev WHERE cur.manager_id = prev.id)

SELECT * FROM st

ID	1	2	3	4	5
NAME	John	Paul	Anna	Peter	Steve
MANAGER_ID		1	1	2	4

RESULT: (‘John’, ‘Paul’, ‘Anna’, ‘Peter, ‘Steve’)

CONNECT BY vs CTEs

Oracle

SELECT name FROM staff
START WITH name = 'John'
CONNECT BY manager_id = PRIOR id

PostgreSQL

WITH RECURSIVE st (id, name, manager_id) AS (SELECT id, name, manager_id FROM staff where name = 'John'

UNION ALL

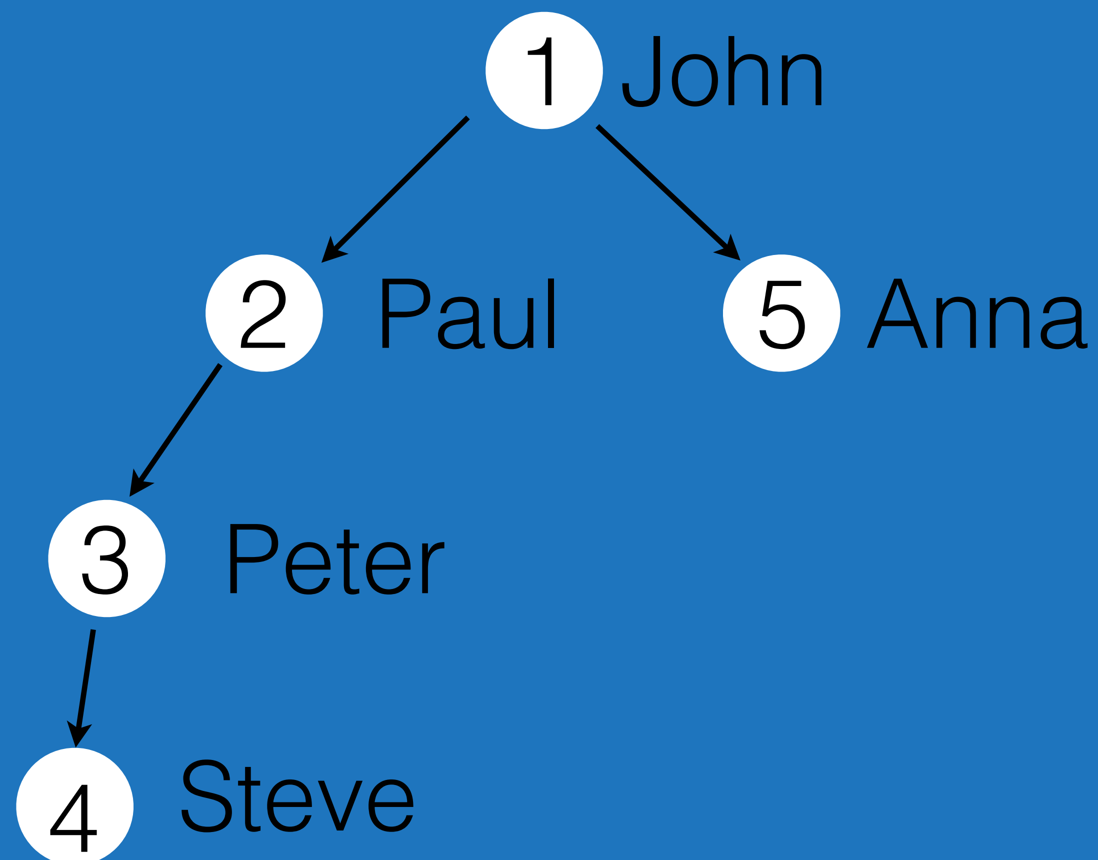
SELECT id, name, manager_id FROM staff cur, st prev WHERE cur.manager_id = prev.id)

SELECT * FROM st

CONNECT BY vs CTEs

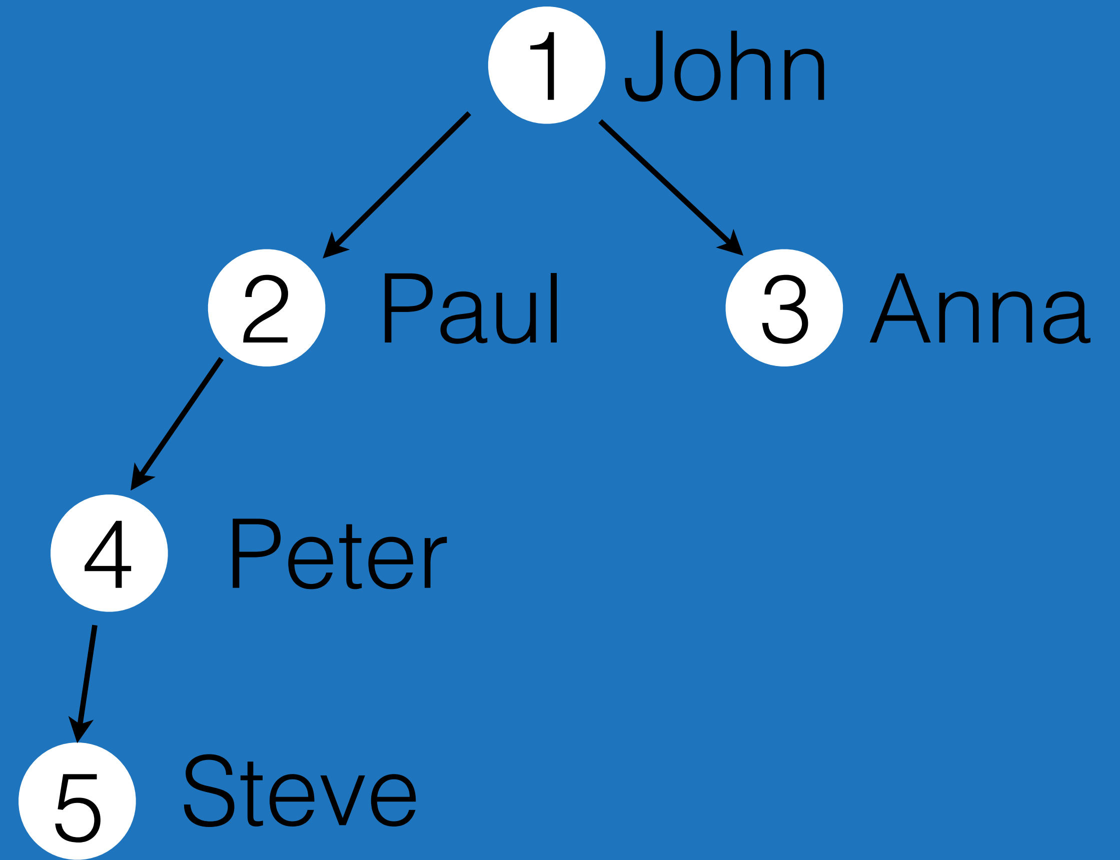
Search order difference

Oracle (depth-first)



(John, Paul, Peter, Steve, Anna)

PostgreSQL (breadth-first)



(John, Paul, Anna, Peter, Steve)

LEVEL and PATH in Oracle

SELECT ID, NAME, LEVEL, SYS_CONNECT_BY_PATH(name, '/') "PATH" FROM staff
START WITH NAME='John' CONNECT BY PRIOR ID = MANAGER_ID

ID	NAME	LEVEL	PATH
1	John	1	/John
2	Paul	2	/John/Paul
4	Peter	3	/John/Paul/Peter
5	Steve	4	/John/Paul/Peter/Steve
3	Anna	2	/John/Anna

LEVEL and PATH in PostgreSQL

WITH RECURSIVE org AS (SELECT id, name, 1 as level, ARRAY[name] AS path FROM staff UNION ALL SELECT next.id, next.name, prev.level + 1 as level, prev.path || next.name as path FROM org prev, staff next WHERE org.id = staff.manager_id)

SELECT id, name, level, '/'||array_to_string(path, '/') as path from org

ID	NAME	LEVEL	PATH
1	John	1	/John
2	Paul	2	/John/Paul
3	Anna	2	/John/Anna
4	Peter	3	/John/Paul/Peter
5	Steve	4	/John/Paul/Peter/ Steve

Matching Oracle's search order

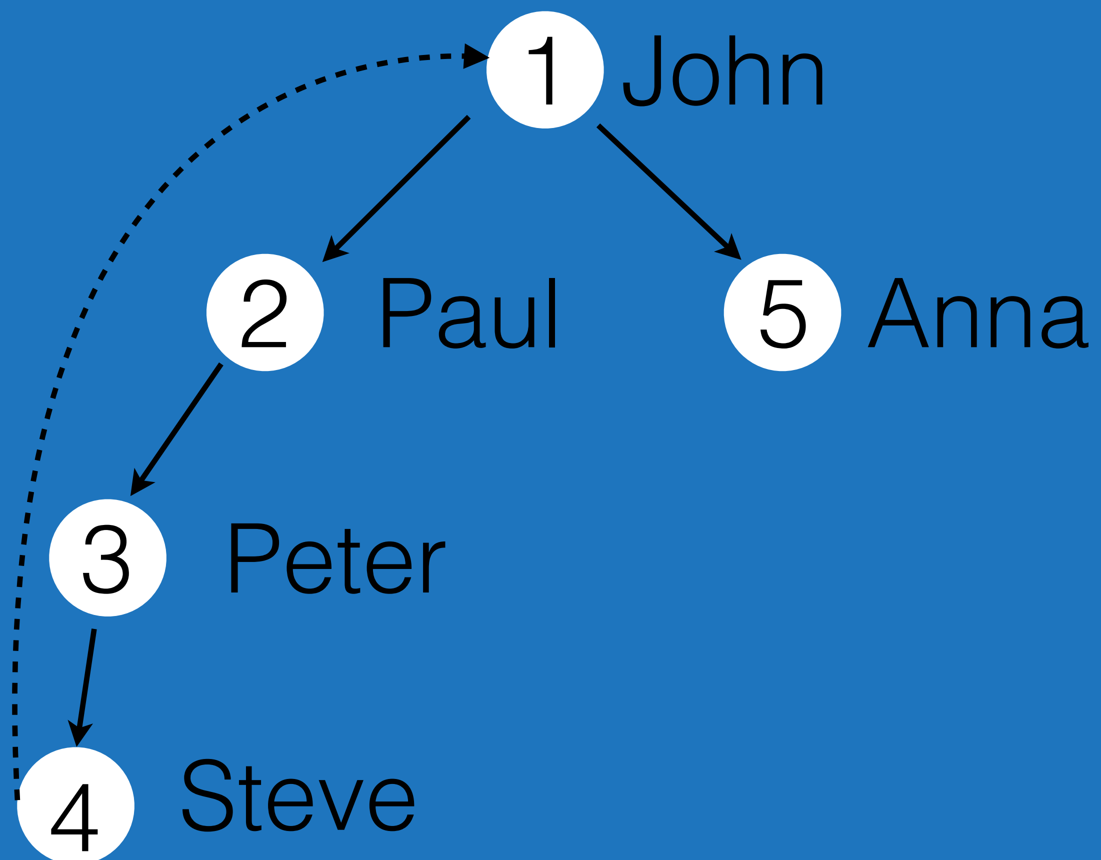
WITH RECURSIVE org AS (SELECT id, name, 1 as level, ARRAY[name] AS path FROM staff UNION ALL SELECT next.id, next.name, prev.level + 1 as level, prev.path || next.name as path FROM org prev, staff next WHERE org.id = staff.manager_id)

SELECT id, name, level, '/'||array_to_string(path, '/') as path from org ORDER BY path

ID	NAME	LEVEL	PATH
1	John	1	/John
2	Paul	2	/John/Paul
4	Peter	3	/John/Paul/Peter
5	Steve	4	/John/Paul/Peter/Steve
3	Anna	2	/John/Anna

Detecting cycles with Oracle

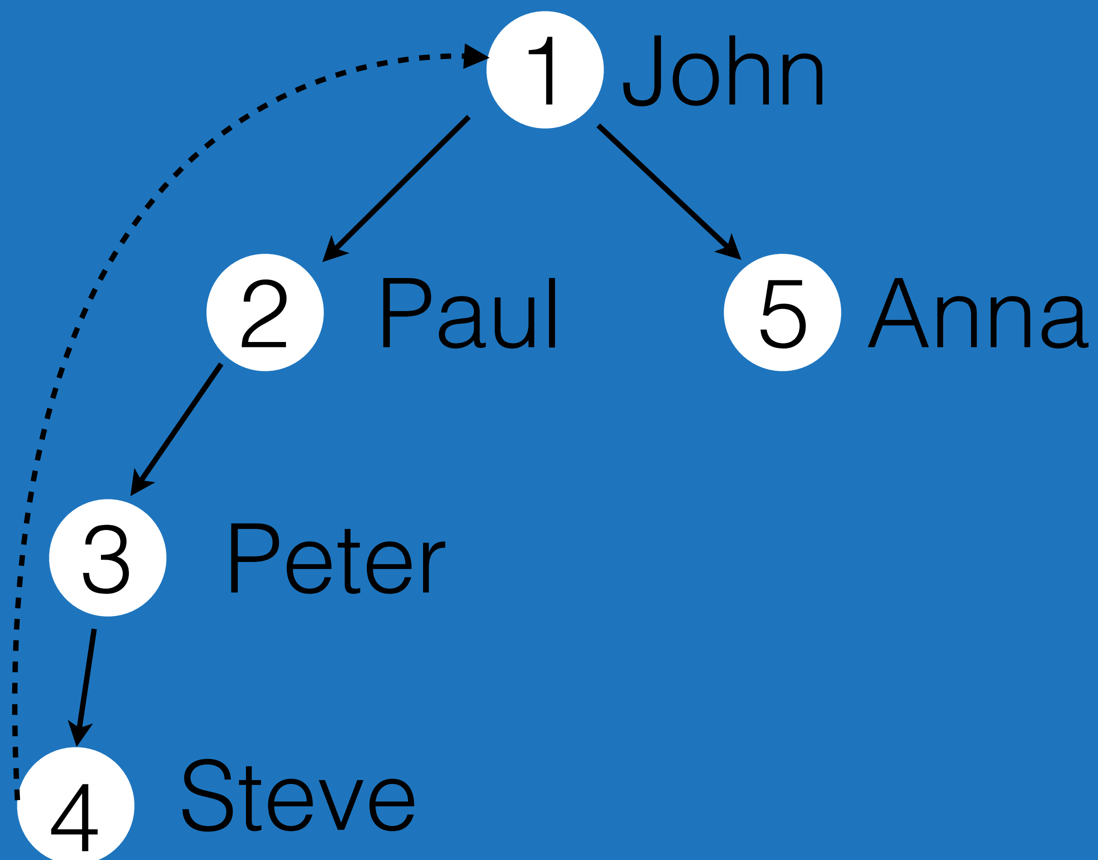
SELECT ID, NAME, LEVEL, SYS_CONNECT_BY_PATH(name, '/') "PATH" FROM staff
START WITH NAME='John' CONNECT BY NOCYCLE PRIOR ID = MANAGER_ID



Detecting cycles with PostgreSQL

WITH RECURSIVE org AS (SELECT id, name, 1 as level, ARRAY[name] AS path, cycle as FALSE FROM staff UNION ALL SELECT next.id, next.name, prev.level + 1 as level, prev.path || next.name as path, next.name = ANY(prev.path) as cycle FROM org prev, staff next WHERE org.id = staff.manager_id) WHERE cycle = FALSE

SELECT id, name, level, '/'||array_to_string(path, '/') as path FROM org WHERE cycle=FALSE



More Oracle CONNECT BY features (not covered)

- CONNECT_BY_ISCYCLE
- CONNECT_BY_ISLEAF
- CONNECT_BY_ROOT
- ORDER SIBLINGS

Translating Oracle functions

- Orafce: orafce.projects.pgfoundry.org
- PL/SQL to PL/pgSQL:

  http://www.postgresql.org/docs/current/static/plpgsql-porting.html

Translating instr

- Orafce
- PostgreSQL documentation
- Corner case:

Oracle	PostgreSQL
SELECT instr(‘foo’, ‘f’, 0) FROM dual RESULT: 0	SELECT instr(‘foo’, ‘f’, 0) FROM dual RESULT: 2

sysdate vs now()

- sysdate - server's timezone
- now() - session's timezone
- implement sysdate as now() at hard-coded timezone in PostgreSQL

Making sure it works

- Hundreds of files, 1 - 10 queries each
- Lack of frameworks for cross-database query testing
- Python to the rescue

Python database drivers

- psycopg2
- cx_Oracle 4.4.1 (with a custom patch)
- 32-bit version to talk to Oracle 8i

Test application workflow

- Establish the database connections
- Read queries from test files
- Run queries against both databases
- Compare results
- Cleanup and exit

Connecting to databases

```
import cx_Oracle
import psycopg2
...
conn_string_pg="dbname=pgdb host=pghost user=slon password=secret"
conn_string_ora=slon/secret@oracledb"
...
def establish_db_connections(self, conn_string_ora, conn_string_pg):
    try:
        self._connora = cx_Oracle.connect(conn_string_ora)
        self._connpg = psycopg2.connect(conn_string_pg)
    except Exception, e:
            if isinstance(e, cx_Oracle.Error):
                raise Exception("Oracle: %s" % (e,))
            elif isinstance(e, psycopg2.Error):
                raise Exception("Postgres: %s" % (e,))
            else:
                raise
```

Reading queries

- Query files parsing
- Variables replacements
- Python is flexible (handles queries embedded in XML easily)

Running queries

```
def get_query_result(self, conn, query, limit=0):
    result = []
    rows = 0
    try:
        cur = conn.cursor()
        cur.execute(str(query))
        for row in cur:
            result.append(row)
            rows += 1
            if rows - limit == 0:
                break
    except Exception, e:
        if isinstance(e, cx_Oracle.Error):
            raise Exception(("Oracle: %s" % (e,)).rstrip('\n\r'))
        elif isinstance(e, psycopg2.Error):
            raise Exception(("Postgres: %s" % (e,)).rstrip('\n\r'))
        else:
            raise
    finally:
        conn.rollback()
    return result
```

Running queries faster

- One thread per database connection
- Asynchronous I/O

Getting result rows from PostgreSQL

- SELECTs are easy
- INSERTs/UPDATEs/DELETEs + RETURNING:

```
INSERT INTO pgconf(year, city) values(2012, 'Prague') RETURNING *;
```

Getting result rows from Oracle

- SELECTs are easy
- INSERTs/UPDATEs/DELETEs - dynamically wrap into anonymous PL/SQL blocks
- INSERT...SELECT is a special case

Anonymous PL/SQL blocks for DML queries example

```
cur = con.cursor()
result=[]
result.append(cur.arrayvar(ora.NUMBER, 1000))
result.append(cur.arrayvar(ora.STRING, 1000))
cur.execute("""
    begin
        insert into pgconf(year,city) values(2012, 'Prague') returning year, city bulk
collect into :1, :2;
    end;""", result)
rows = zip(*(x.getvalue() for x in result))
cur.close()
```

Getting table information from Oracle

```
SELECT

TABLE_NAME, COLUMN_NAME,
DATA_TYPE, DATA_PRECISION,
DATA_SCALE,
CHAR_COL_DECL_LENGTH

FROM  ALL_TAB_COLUMNS

WHERE TABLE_NAME='pgconf'
ORDER BY COLUMN_ID ASC
```

Unsupported features by PL/SQL in 8i

- Scalar subselects
- LONG RAW columns
- CASE...WHEN blocks

Questions?

Twitter: @alexeyklyukin

Email: alexk@commandprompt.com

References

- http://ora2pg.darold.net/index.html - Ora2pg home page
- http://keithf4.com/oracle_fdw - using Oracle FDW to migrate from 8i
- http://www.postgresql.org/docs/8.3/interactive/plpgsql-porting.html - PostgreSQL documentation chapter on porting PL/SQL code
- http://orafce.projects.postgresql.org/ - Orafce home page
- http://cx-oracle.sourceforge.net/html/index.html - cx_Oracle documentation
- http://www.initd.org/psycopg/docs/ - psycopg2 documentation
- http://code.google.com/p/python-sqlparse/ - Python SQL parser library
- http://docs.python.org/library/markup.html - python libraries to work with structured data markup

Thank you!

Feedback: 2012.pgconf.eu/feedback/