Continuous Queries In Oracle

A. Witkowski, S. Bellamkonda, H. Li, V. Liang, L. Sheng, Q. Smith,
S. Subramanian, J. Terry, T. Yu

Oracle Corporation

ORACLE

Continuous Query – Problem Statement

- Continuous Query – what is needed in RDBMS
  - User's queries define interesting states (negative balance)
  - Monitor the change of state (alert if balance goes negative)
  - Sources are the changes to the relational tables
  - State change, Query Delta, fundamental to CQ
- CQ doesn't exist in RDBMS. Users have to poll data.
- Polling mode is inconvenient and non performant
  - Involves executing queries over all data.
  - Returns the same answers if no state change
  - There can be thousands CQ and RDBMS can optimize

Static Query – Example Problem Statement

Get people whose sum of transactions drops below 0

```
Query Q:
SELECT account, sum(amt)
FROM t
GROUP BY account
HAVING sum(amt) < 0
```

TRANSACTION T

Acct	time	Amt
Andy	11-15-05	+100
Andy	11-16-05	+100
Joe	11-17-05	+200
Andy	11-18-05	-200
Joe	11-19-05	+100
Andy	11-20-05	-200

⟹

Account	sum(amt)
_	

⟹

Account	sum(amt)
Andy	-200

Today users have to run the query periodically on all data.
Query may return the same data.

4

Continuous Query – Has Query Delta Semantics

Get people whose sum of transactions drops below 0

TRANSACTION T

Acct	time	Amt
Andy	11-15-05	+100
Andy	11-16-05	+100
Joe	11-17-05	+200
Andy	11-18-05	-200

query delta

⟹

_	_

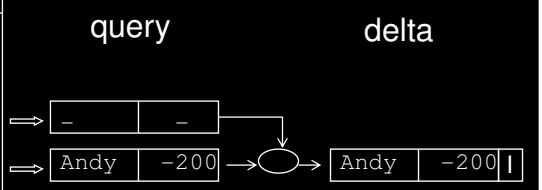
5

Continuous Query – Has Query Delta Semantics

Get people whose sum of transactions drops below 0

TRANSACTION T

Acct	time	Amt
Andy	11-15-05	+100
Andy	11-16-05	+100
Joe	11-17-05	+200
Andy	11-18-05	-200
Joe	11-19-05	+100
Andy	11-20-05	-200



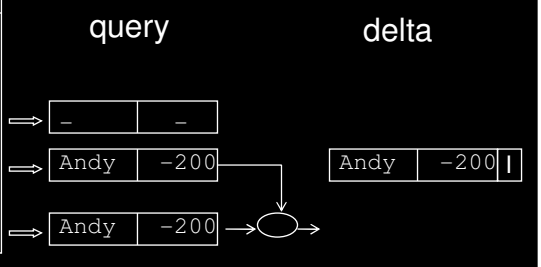
6

Continuous Query – Has Query Delta Semantics

Get people whose sum of transactions drops below 0

TRANSACTION T

Acct	time	Amt
Andy	11-15-05	+100
Andy	11-16-05	+100
Joe	11-17-05	+200
Andy	11-18-05	-200
Joe	11-19-05	+100
Andy	11-20-05	-200
Joe	11-21-05	-100
Joe	11-22-05	-100



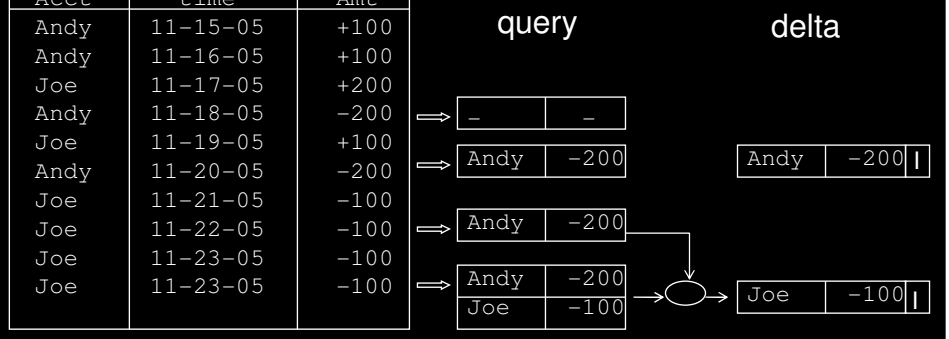
7

Continuous Query – Has Query Delta Semantics

Get people whose sum of transactions drops below 0

TRANSACTION T

Acct	time	Amt
Andy	11-15-05	+100
Andy	11-16-05	+100
Joe	11-17-05	+200
Andy	11-18-05	-200
Joe	11-19-05	+100
Andy	11-20-05	-200
Joe	11-21-05	-100
Joe	11-22-05	-100
Joe	11-23-05	-100
Joe	11-23-05	-100

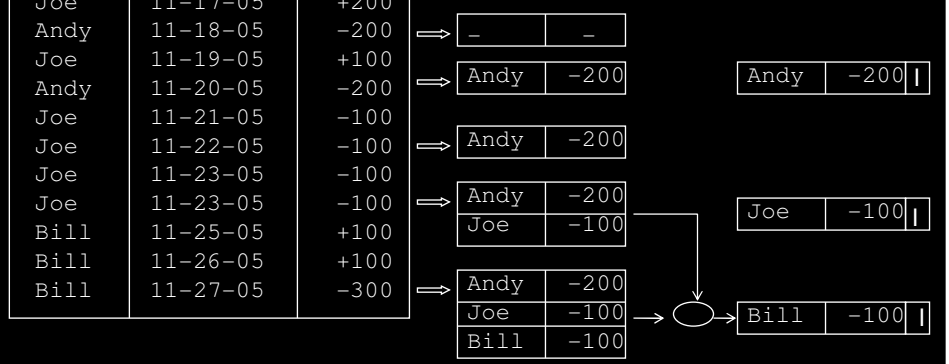


Continuous Query – Has Query Delta Semantics

Get people whose sum of transactions drops below 0

TRANSACTION T

Acct	time	Amt
Andy	11-15-05	+100
Andy	11-16-05	+100
Joe	11-17-05	+200
Andy	11-18-05	-200
Joe	11-19-05	+100
Andy	11-20-05	-200
Joe	11-21-05	-100
Joe	11-22-05	-100
Joe	11-23-05	-100
Joe	11-23-05	-100
Bill	11-25-05	+100
Bill	11-26-05	+100
Bill	11-27-05	-300

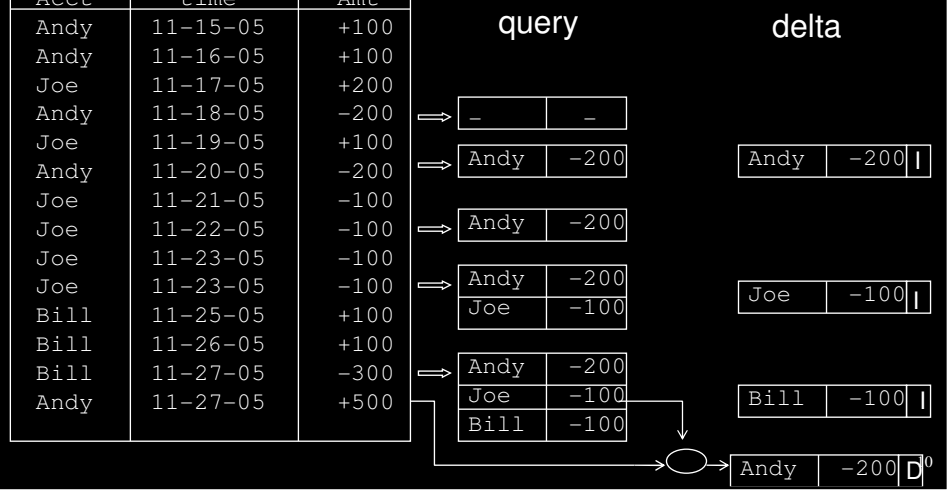


Continuous Query – Has Query Delta Semantics

Get people whose sum of transactions drops below 0

TRANSACTION T

Acct	time	Amt
Andy	11-15-05	+100
Andy	11-16-05	+100
Joe	11-17-05	+200
Andy	11-18-05	-200
Joe	11-19-05	+100
Andy	11-20-05	-200
Joe	11-21-05	-100
Joe	11-22-05	-100
Joe	11-23-05	-100
Joe	11-23-05	-100
Bill	11-25-05	+100
Bill	11-26-05	+100
Bill	11-27-05	-300
Andy	11-27-05	+500



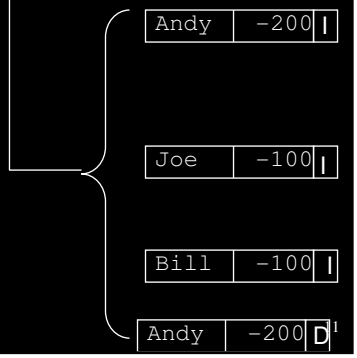
Continuous Query – Has Query Delta Semantics

Get people whose sum of transactions drops below 0

TRANSACTION T

Acct	time	Amt
Andy	11-15-05	+100
Andy	11-16-05	+100
Joe	11-17-05	+200
Andy	11-18-05	-200
Joe	11-19-05	+100
Andy	11-20-05	-200
Joe	11-21-05	-100
Joe	11-22-05	-100
Joe	11-23-05	-100
Joe	11-23-05	-100
Bill	11-25-05	+100
Bill	11-26-05	+100
Bill	11-27-05	-300
Andy	11-27-05	+500

```
CREATE CQ AS DESTIN AQ
SELECT account, sum(amt)
FROM t
GROUP BY account
HAVING sum(amt) < 0
```

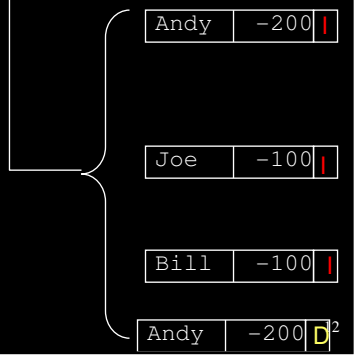


Continuous Query – Has Query Delta Semantics

Get people whose sum of transactions drops below 0

```
CREATE CQ AS DESTIN AQ
SELECT account, sum(amt)
FROM t
GROUP BY account
HAVING sum(amt) < 0
```

INSERT DELTA
when data appears in query result
DELETE DELTA
when data disappears from it
UPDATE DELTA
when data changes in query result



Query Delta –Language Bindings

```
-- Inform if balance goes below 0
CREATE CONTINUOUS QUERY negative_balance_cq
  PRIMARY KEY (acct)
  COMPUTE TRANSACTIONAL
          INSERT DELETE DELTA
          ON COMMIT
  DESTINATION dest_table
AS
  SELECT acct, sum(amt) bal, delta_marker()
  FORM transaction
  GROUP BY acct
  HAVING sum(amt) < 0
```

Primary Key
Type of Delta
Part of Delta
Frequency of computation
Destination
Delta Marker
Defining query

13

Transactional and Compressed Delta

Acct	Time	Amt
Andy	07-01-06	100
Andy	07-02-06	-200
Andy	07-03-06	300
Mark	07-07-06	400

Acct	Time	Amt	Mark

```
CREATE CONTINUOUS QUERY negative_balance_cq
  PRIMARY KEY (acct)
  COMPUTE COMPRESSED DELTA EVERY INTERVAL '7' DAYS
SELECT acct, sum(amt) bal, delta_marker() mark
FORM transaction_tbl
GROUP BY acct
HAVING sum(amt) < 0)
```

14

Transactional and Compressed Delta

Acct	Time	Amt
Andy	07-01-06	100
Andy	07-02-06	-200
Andy	07-03-06	300
Mark	07-07-06	400

Acct	Time	Amt	mark
Andy	07-02-06	-100	I
Andy	07-03-06	-100	D

```
CREATE CONTINUOUS QUERY negative_balance_cq
  PRIMARY KEY (acct)
  COMPUTE TRANSACTIONAL DELTA EVERY INTERVAL '7' DAYS
SELECT acct, sum(amt) bal, delta_marker() mark
FORM transaction_tbl
GROUP BY acct
HAVING sum(amt) < 0)
```

15

CQ – Description (1)

- Continuous Query Delta – a new SQL object with a query
  - Persisted declaration analogous to familiar View syntax
- Query Delta – Computes continual changes to query
  - INSERT, DELETE, UPDATE deltas
  - Deltas are Transaction Consistent (I.e., we see committed changes)
  - Compressed and Transactional Deltas
- Sources of CQ
  - DML Changes to Relational Tables
  - Changes logged to mv logs. One log per base table stores before and after images of row changes
- Destination of CQ
  - Tables – will record the history of all changes (auditing)
  - Triggers – procedural processing of events
  - Oracle Queues - APIs to de-queue asynchronously
  - Callbacks – Java or C procedures called when delta produced

16

CQ – Description (2)

- SQL supporting functions
  - Cq_delta_maker, cq_old_value
  - Cq_time, cq_commit_time
- CQ computation. How
  - How: Asynchronous. DML commits, and then activate CQs.
- CQ computation. When
  - Commit on Sources (ON COMMIT)
  - Periodic (START '01-01-2007' WITH PERIOD 1 DAY)
  - ON DML to sources (ON INSERT OR UPDATE TO t)
- Query Shapes for CQ
  - CQJ – queries with (semi, outer, inner) joins only
  - CQAJ – queries with Anti-Joins for non events
  - CQA – queries with aggregation and joins
  - CQW – queries with window functions

17

Continuous Queries with Joins: General Computation

- Changes to tables, Δ, logged in their logs (logs are tables).
- Consider CQJ Q = T >< S, e.g., T.c = S.c
- Q image before `pre(Q)` and after `pst(Q)`

```
Δ(Q) = pst(Q) - pre(Q)
     = pst(T) >< pst(S) - pre(T) >< pre(S)
     = (pre(T)+ Δ(T))><(pre(S)+ Δ(S)) - pre(T) >< pre(S)
(1)  = pre(T) >< Δ(S) + Δ(T) >< pst(S)
(2)  = pst(T) >< Δ(S) + Δ(T) >< pre(S)
(3)  = pst(T) >< Δ(S) + Δ(T) >< pst(S) - Δ(T) >< Δ(S)
```

- Delta expressions similar to MV refresh. N^2 joins
- How to obtain pre-image.
- Which form to use (1, 2, or 3)? Any other

18

Continuous Queries with Joins (CQJ): Optimizations

- Refresh expressions use recursive SQL
- Obtaining Pre-image. SQL vs application of undo

```
SQL: pre(T) = SELECT * FROM T WHERE T.rowid NOT IN
                    (SELECT rowid FROM clog_t)

UNDO: pre(T)= SELECT * FROM T AS OF pre_image_time
```

- Which form to use for `Δ(Q)`

```
(1) = pre(T) >< Δ(S) + Δ(T) >< pst(S)
(2) = pst(T) >< Δ(S) + Δ(T) >< pre(S)
(3) = pst(T) >< Δ(S) + Δ(T) >< pst(S) - Δ(T) >< Δ(S)
```

- Not 3 since requires MINUS & complex for N tables.
- (1) if card(T) < card(S) and (2) otherwise

19

Continuous Queries with Joins. FK-PK optimization

Consider Q = T >< S and enforced T.fk = S.pk

$\Delta(Q) = pre(T) >< \Delta(S) + \Delta(T) >< pst(S)$

$= pst(T) >< \Delta(S) + \Delta(T) >< pre(S)$

Inserts. $\Delta(S)$ does not join with $pre(T)$

$\Delta_I(Q) = \Delta(T) >< pst(S)$

Deletes. Deleting joining rows from S deletes from T:

$\Delta_D(Q) = \Delta(T) >< pre(S) = \Delta(T) >< (pst(S) + \Delta(S))$

Mix DML.

$\Delta(Q) = \Delta_D(Q) + \Delta_I(Q)$

For CQJ with N tables reduces number for joins N^2 to 2*N.

20

Continuous Queries with Joins. FK-PK units (1)

CQJ: alert if Mary buys shoes again:

orderline

item	Price	oid
Shoes	100	200
Pants	50	200

orders

oid	customer	Date
200	Andy	01-01-07

```
CREATE CONTINOUS QUERY mary_shoes
  DESTINATION dest
  COMPUTE ON COMMIT
SELECT item, o.oid, date
FROM orderline ol, orders O
WHERE ol.oid=o.oid AND customer = 'Mary' AND item=shoes
```

21

Continuous Queries with Joins. FK-PK units (1)

CQJ: alert if Mary buys shoes again:

orderline

item	Price	oid
Shoes	100	200
Pants	50	200
Shirt	30	300
Tie	15	300

orders

oid	customer	Date
200	Andy	01-01-07
300	Bill	01-02-07

22

Continuous Queries with Joins. FK-PK units (1)

CQJ: alert if Mary buys shoes again:

orderline

item	Price	oid
Shoes	100	200
Pants	50	200
Shirt	30	300
Tie	15	300
Blouse	80	400
Shoes	60	400

orders

oid	customer	Date
200	Andy	01-01-07
300	Bill	01-02-07
400	Mary	01-02-07

23

Continuous Queries with Joins. FK-PK units (1)

CQJ: alert if Mary buys shoes again:

orderline

item	Price	oid
Shoes	100	200
Pants	50	200
Shirt	30	300
Tie	15	300
Blouse	80	400
Shoes	60	400

orders

oid	customer	Date
200	Andy	01-01-07
300	Bill	01-02-07
400	Mary	01-02-07

If insert transaction arrive in FK-PK units then:

$\Delta(Q) = \Delta(\text{orderline}) >< \Delta(\text{orders})$

24

Continuous Queries with Joins. FK-PK units (1)

CQJ: alert if Mary buys shoes again:

orderline

item	Price	oid
Shoes	100	200
Pants	50	200
Shirt	30	300
Tie	15	300
Blouse	80	400
Shoes	60	400

orders

oid	customer	Date
200	Andy	01-01-07
300	Bill	01-02-07
400	Mary	01-02-07

If insert transaction arrive in FK-PK units then:

$\Delta(Q) = \Delta(\text{orderline}) >< \Delta(\text{orders})$

What if not. Δ on FK (orderline) joins with pst(PK)

25

Continuous Queries with Joins. FK-PK units (2)

Discover anti-join tuples in

```
Δ(orderline) LEFT OUTER JOIN Δ(orders)
```

Store in temp_table and join them with pst(orders).

```
INSERT
  WHEN aj_mark = 1 INTO anti_join(item, oid)
  WHEN aj_mark = 0 INTO dest(item, o.oid, date)
SELECT item, o.oid, aj_mark
FROM Δ(orderline ol)LOJ Δ(orders o) ON ol.oid=o.oid
```

Check if anti_join empty

```
SELECT COUNT(*) FROM anti_join
```

And if not, add anti-join tuples to dest table.

```
INSERT INTO dest(item, o.oid, date)
SELECT item, o.oid
FROM anti_join aj JOIN pst(orders) o ON aj.oid=o.oid
```

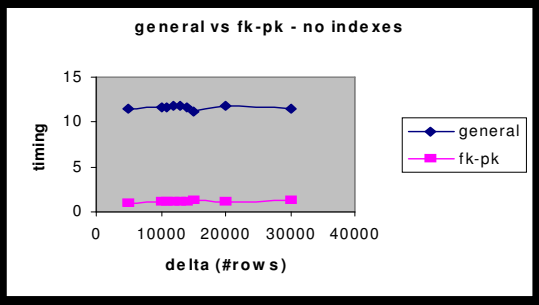
26

Performance Evaluation

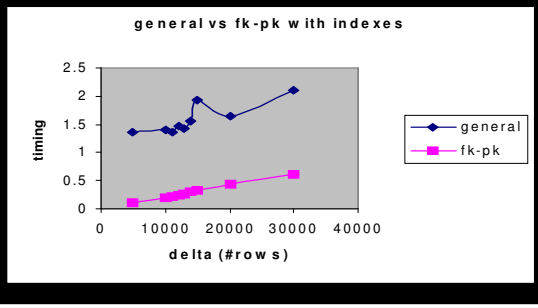
- E-store application with
  - Orderline - 10M rows
  - Orders – 1 M rows
  - customer - 100K rows
- Customers typically buy 10 items per order
- Experiments:
  - Changed refresh times, i.e. size of the deltas
  - Consider tables with and without indexes
  - Consider optimizations FK-PK, FK-PK units, undo

27

Performance of general vs FK-PK CQ computation



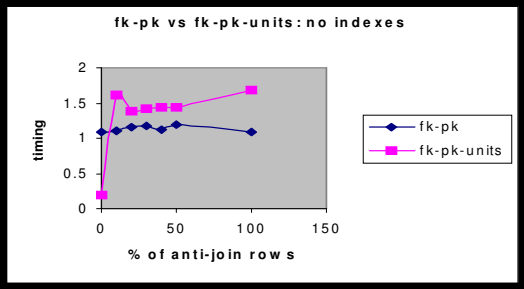
Hash Join
Useful 4 large Δ
10x faster



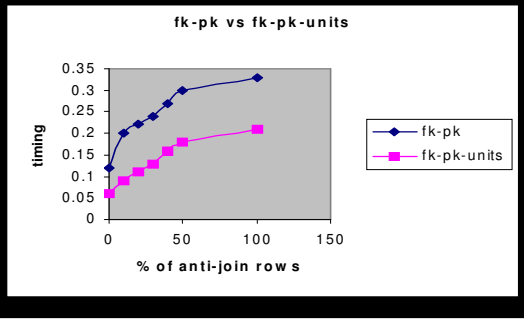
Nested Loop Joi
Useful 4 small Δ
4x faster

28

Performance of FK-PK vs FK-PK units computation (1)



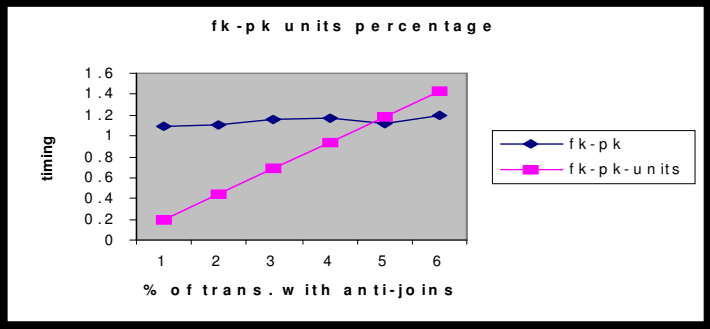
Hash Join
Δ = 200 orders
5x faster – 20% slowe



Indexes
Nested Loop Join
Δ = 200 orders
1.5x – 2x faster

29

Performance of FK-PK vs FK-PK units computation (2)



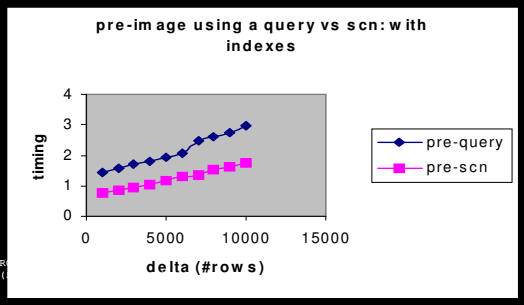
For Transactions not in FK-PK units, anti-join rows are 50%

```
|(orderline) anti-join (orders)| = 0.5*|(orderline) LOJ (orders)|
```

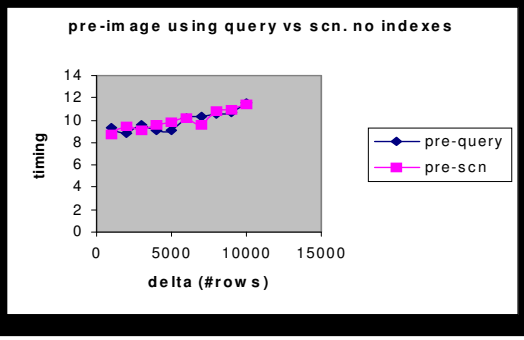
If we have less than 80% with anti-join rows, FK-PK-units optimization is better than general FK-PK.

30

Computing pre image with SQL vs undo application



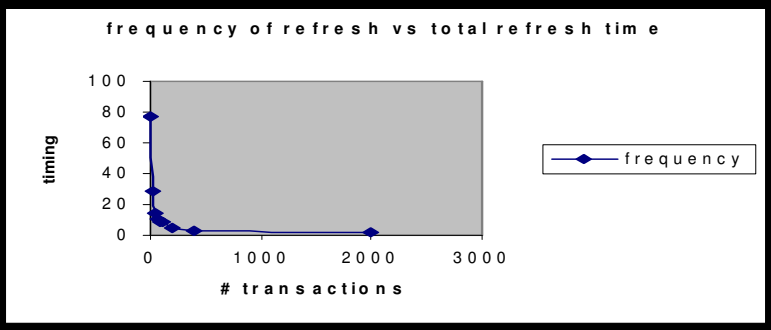

```
SELECT * FROM orders
WHERE rowid NOT IN
   (SELECT rowid
    FROM dlt(orders))
VS:
SELECT *
FROM orders AS OF scn
```



Useful if refresh time small. Benefit up to 1.5x faster
Overhead of hash join
kills benefit of scn

31

Frequency of Refresh



- FK-PK optimized refresh expressions
- 10 orders per transaction. Processed total of 7000 transactions
- Refresh varies from 10-2000 transactions
- If refresh every 10 transactions, it is 35x slower that a single refresh
- After some threshold, 100 transactions, frequency has little

32

Conclusions and Future Work

- This work
  - Formal definition of CQ based on query delta
  - New algorithms for MV and CQ refresh
  - New CQ shapes – window functions
- Future Work
  - Multi-query optimizations
  - Cover more query shapes in CQ
  - Incorporate pattern recognition in sequences of rows (ANSI SQL work with IBM, Streambase, Coral8)
  - Incorporate stream (CQL) semantics (ANSI SQL work with IBM, Streambase, Coral8)

33