Improving SQL Query Performance - Optimizer Directives

Bingjie Miao
IBM

A16 (83)
Wednesday, April 30, 2008 • 2:10 p.m. – 3:10 p.m.

2008 IIUG Informix Conference

www.iiug.org

Agenda

- Introducing optimizer directives
- Types of optimizer directives
- How to use optimizer directives
- External optimizer directives

Agenda

- Introducing optimizer directives
- Types of optimizer directives
- How to use optimizer directives
- External optimizer directives

IDS optimizer

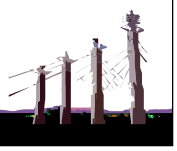
- Cost-based optimizer
- Chooses “best” query execution plan based on estimated cost for each execution plan
- Utilizes statistics on tables and indexes gathered via “update statistics” command
- Chosen query execution plan can be generated in explain file

Optimizer directives

- Optimizer directives are comments that instruct the query optimizer how to execute a query
  - direct query optimizer to use desired access path, join method, join ordering, etc. to achieve desired query execution plan

Why optimizer directives?

- Query optimizer makes mistakes
  - lack necessary statistics
  - out-dated statistics
  - limitations in query optimizer
  - defects in query optimizer
- Allows user to influence query optimization process to achieve desired query execution plan



Ways to influence query optimizer

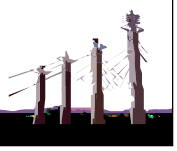
- ONCONFIG parameters
  - instance wide
  - OPT_GOAL, OPTCOMPIND, DIRECTIVES
- Session environments
  - session wide
  - set optimization high/low
  - set environment optcompind ‘0’/’1’/’2’
- Optimizer directives
  - affect a single query

An example

```
SELECT {+INDEX(customer)} *
FROM customer, orders
WHERE customer.customer_num =
  orders.customer_num
  AND customer.state = "NY"
```

An example – explain file

```
SELECT {+INDEX(customer)} * FROM customer, orders
WHERE customer.customer_num = orders.customer_num AND customer.state = "NY"

DIRECTIVES FOLLOWED:
INDEX ( customer )
DIRECTIVES NOT FOLLOWED:

Estimated Cost: 7
Estimated # of Rows Returned: 2

  1) usr1.customer: INDEX PATH

       Filters: usr1.customer.state = 'NY'

    (1) Index Keys: customer_num   (Serial, fragments: ALL)

  2) usr1.orders: INDEX PATH

    (1) Index Keys: customer_num   (Serial, fragments: ALL)
        Lower Index Filter: usr1.customer.customer_num = usr1.orders.customer_num
NESTED LOOP JOIN
```



Agenda

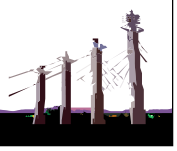
- Introducing optimizer directives
- Types of optimizer directives
- How to use optimizer directives
- External optimizer directives

Types of optimizer directives

- Access method directives
- Join method directives
- Join order directives
- Optimization goal directives
- Explain directives

Access method directives

- FULL
- AVOID_FULL
- INDEX
- AVOID_INDEX
- INDEX_SJ
- AVOID_INDEX_SJ

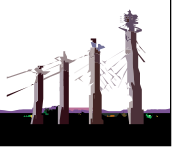


FULL

- FULL (tabname)

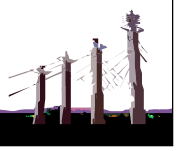
  directs the query optimizer to favor a full table scan path (sequential scan path) for the specified table

```
SELECT {+ FULL(customer)} *
FROM customer
WHERE customer.state != “NV”
```



FULL – explain file

```
SELECT {+ FULL(customer)} *
FROM customer
WHERE customer.state != "NV"

DIRECTIVES FOLLOWED:
FULL ( customer )
DIRECTIVES NOT FOLLOWED:

Estimated Cost: 2
Estimated # of Rows Returned: 25

  1) usr1.customer: SEQUENTIAL SCAN

      Filters: usr1.customer.state != 'NV'
```



AVOID_FULL

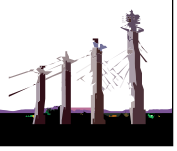
- AVOID_FULL (tabname)

  directs the query optimizer to avoid a full table scan path (sequential scan path) for the specified table

```
SELECT {+ AVOID_FULL(customer)} *
FROM customer
WHERE customer.customer_num
        between 100 and 110
```

AVOID_FULL – explain file

```
SELECT {+ AVOID_FULL(customer)} *
FROM customer
WHERE customer.customer_num between 100 and 110

DIRECTIVES FOLLOWED:
AVOID_FULL ( customer )
DIRECTIVES NOT FOLLOWED:

Estimated Cost: 2
Estimated # of Rows Returned: 9

 1) usr1.customer: INDEX PATH

   (1) Index Keys: customer_num   (Serial, fragments: ALL)
       Lower Index Filter: usr1.customer.customer_num >= 100
       Upper Index Filter: usr1.customer.customer_num <= 110
```

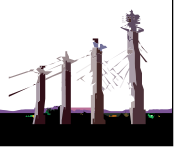


INDEX

- INDEX (tabname)
- INDEX (tabname index_list)

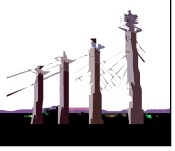
  Directs the query optimizer to use an index path for the specified table (choose from indexes in the specified list, or all indexes if list not specified)

  The best index is chosen from all candidates



INDEX - examples

```
SELECT {+ INDEX (customer)} *
FROM customer WHERE ……

SELECT {+ INDEX (customer cidx1)} *
FROM customer WHERE ……

SELECT {+ INDEX (customer cidx1 cidx2)} *
FROM customer WHERE ……
```

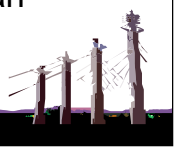


INDEX – explain file

```
SELECT {+ INDEX (customer cidx1 cidx2)} *
FROM customer WHERE state = "CA"

DIRECTIVES FOLLOWED:
INDEX ( customer cidx1 cidx2 )
DIRECTIVES NOT FOLLOWED:

Estimated Cost: 30
Estimated # of Rows Returned: 28

  1) usr1.customer: INDEX PATH

    (1) Index Keys: state city   (Serial, fragments: ALL)
        Lower Index Filter: usr1.cust1.state = 'CA'
```

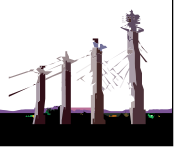
AVOID_INDEX

- AVOID_INDEX (tabname index_list)
  Directs the query optimizer to avoid considering using an index in the specified list in accessing the specified table (indexes not specified in the list can still be considered)
  If all available indexes are specified in the list, then this will effectively force full table scan



AVOID_INDEX - example

```
SELECT {+ AVOID_INDEX(customer cidx1)} *
FROM customer WHERE ……

SELECT
    {+ AVOID_INDEX (customer cidx1 cidx2)} *
FROM customer WHERE ……
```



AVOID_INDEX – explain file

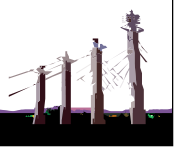
select **{+ AVOID_INDEX (cust1 cidx1 cidx2)}** *
from cust1 where state != ' CA'

DIRECTIVES FOLLOWED:
AVOID_INDEX (cust1 cidx1 cidx2)
DIRECTIVES NOT FOLLOWED:

Estimated Cost: 2
Estimated # of Rows Returned: 1

1) usr1.cust1: SEQUENTIAL SCAN

Filters: usr1.cust1.state != ' CA'



INDEX_SJ / AVOID_INDEX_SJ

- INDEX_SJ (tabname idx_list)
- AVOID_INDEX_SJ (tabname idx_list)

  Directs the query optimizer to favor / avoid index self join path using any index in the specified index list for the specified table

  Index self join is a new index access method introduced in IDS 11.10 (Cheetah)

Join method directives

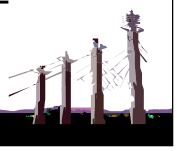
- USE_NL
- AVOID_NL
- USE_HASH
- AVOID_HASH

USE_NL

- USE_NL (tabname)

  Directs the query optimizer to use nested-loop join for the specified table

```
SELECT {+ USE_NL (cust_calls)} *
FROM customer c, cust_calls u
WHERE c.customer_num = u.customer_num
```



USE_NL – explain file

SELECT **{+ USE_NL (cust_calls)}** * FROM customer c, cust_calls u
WHERE c.customer_num = u.customer_num

DIRECTIVES FOLLOWED:
USE_NL (cust_calls)
DIRECTIVES NOT FOLLOWED:

Estimated Cost: 9
Estimated # of Rows Returned: 7

1) usr1.c: SEQUENTIAL SCAN

2) usr1.u: INDEX PATH

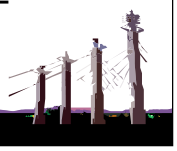
(1) Index Keys: customer_num call_dtime (Serial, fragments: ALL)
Lower Index Filter: usr1.c.customer_num = usr1.u.customer_num
NESTED LOOP JOIN

AVOID_NL

- AVOID_NL (tabname)

  Directs the query optimizer to avoid nested-loop join for the specified table

```
SELECT {+ AVOID_NL (cust_calls)} *
FROM customer c, cust_calls u
WHERE c.customer_num = u.customer_num
```



AVOID_NL – explain file

SELECT **{+ AVOID_NL (cust_calls)}** * FROM customer c, cust_calls u
WHERE c.customer_num = u.customer_num

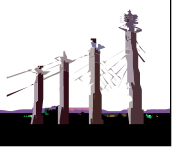
DIRECTIVES FOLLOWED:
AVOID_NL (cust_calls)
DIRECTIVES NOT FOLLOWED:

Estimated Cost: 7
Estimated # of Rows Returned: 7

1) usr1.u: SEQUENTIAL SCAN

2) usr1.c: INDEX PATH

(1) Index Keys: customer_num (Serial, fragments: ALL)
Lower Index Filter: usr1.c.customer_num = usr1.u.customer_num
NESTED LOOP JOIN



USE_HASH

- USE_HASH (tabname)
- USE_HASH (tabname/BUILD)
- USE_HASH (tabname/PROBE)

  Directs the query optimizer to use hash join (on build or probe side if specified) for the specified table

```
SELECT {+ USE_HASH (cust_calls/BUILD)} *
FROM customer c, cust_calls u
WHERE c.customer_num = u.customer_num
```

USE_HASH – explain file

SELECT **{+ USE_HASH (cust_calls/BUILD)}** * FROM customer c, cust_calls u
WHERE c.customer_num = u.customer_num

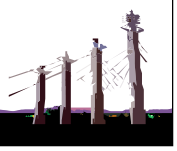
DIRECTIVES FOLLOWED:
USE_HASH (cust_calls/BUILD)
DIRECTIVES NOT FOLLOWED:

Estimated Cost: 12
Estimated # of Rows Returned: 7

1) usr1.c: SEQUENTIAL SCAN

2) usr1.u: SEQUENTIAL SCAN

DYNAMIC HASH JOIN
Dynamic Hash Filters: usr1.c.customer_num = usr1.u.customer_num

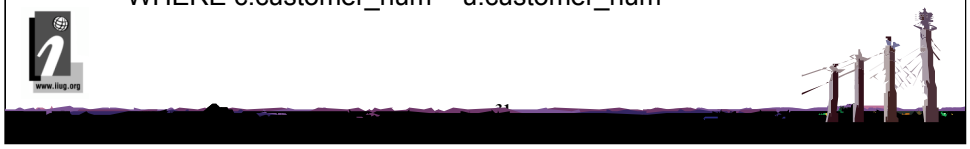


AVOID_HASH

- AVOID_HASH (tabname)
- AVOID_HASH (tabname/BUILD)
- AVOID_HASH (tabname/PROBE)

  Directs the query optimizer to avoid hash join (on build or probe side if specified) for the specified table

```
SELECT {+ AVOID_HASH (cust_calls/PROBE)} *
FROM customer c, cust_calls u
WHERE c.customer_num = u.customer_num
```



AVOID_HASH – explain file

SELECT **{+ AVOID_HASH (cust_calls/PROBE)}** * FROM customer c, cust_calls u
WHERE c.customer_num = u.customer_num

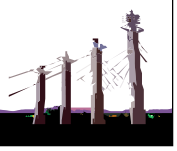
DIRECTIVES FOLLOWED:
AVOID_HASH (cust_calls/PROBE)
DIRECTIVES NOT FOLLOWED:

Estimated Cost: 7
Estimated # of Rows Returned: 7

1) usr1.u: SEQUENTIAL SCAN

2) usr1.c: INDEX PATH

(1) Index Keys: customer_num (Serial, fragments: ALL)
Lower Index Filter: usr1.c.customer_num = usr1.u.customer_num
NESTED LOOP JOIN

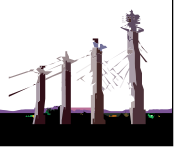


Join order directives

- ORDERED

  Direct query optimizer to join tables in the same order as specified in the query

```
SELECT {+ ORDERED} *
FROM customer c, cust_calls u
WHERE c.customer_num = u.customer_num
```



ORDERED – explain file

SELECT **{+ ORDERED}** * FROM customer c, cust_calls u
WHERE c.customer_num = u.customer_num

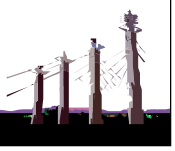
DIRECTIVES FOLLOWED:
ORDERED
DIRECTIVES NOT FOLLOWED:

Estimated Cost: 9
Estimated # of Rows Returned: 7

1) usr1.c: SEQUENTIAL SCAN

2) usr1.u: INDEX PATH

(1) Index Keys: customer_num call_dtime (Serial, fragments: ALL)
Lower Index Filter: usr1.c.customer_num = usr1.u.customer_num
NESTED LOOP JOIN



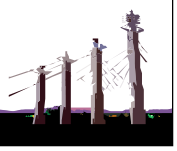
Optimization goal directives

- ALL_ROWS

  Direct query optimizer to choose query execution plan that returns all result rows quickly (this is default behavior)

- FIRST_ROWS

  Direct query optimizer to choose query execution plan that returns first set of result rows quickly



FIRST_ROWS example

- FIRST_ROWS optimization tends to avoid paths that must consume all input rows before producing first output row, such as hash join, or sort

```
SELECT {+ FIRST_ROWS} first 2 *
FROM customer c, cust_calls u
WHERE c.customer_num = u.customer_num
```

FIRST_ROWS – explain file

SELECT **{+ FIRST_ROWS}** first 2 * FROM customer c, cust_calls u
WHERE c.customer_num = u.customer_num

DIRECTIVES FOLLOWED:
FIRST_ROWS
DIRECTIVES NOT FOLLOWED:

Estimated Cost: 7
Estimated # of Rows Returned: 7

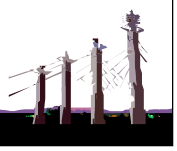
1) usr1.u: SEQUENTIAL SCAN

2) usr1.c: INDEX PATH

(1) Index Keys: customer_num (Serial, fragments: ALL)
Lower Index Filter: usr1.c.customer_num = usr1.u.customer_num
NESTED LOOP JOIN

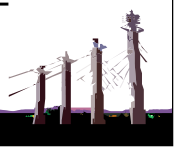
Explain directives

- EXPLAIN
  - Direct query optimizer to generate explain information for the query
  - Semantically equivalent to “set explain on”
- EXPLAIN AVOID_EXECUTE
  - Similar to EXPLAIN, but avoids execution of the query plan (generate explain only)
  - Semantically equivalent to
    - “set explain on avoid_execute”



Explain - example

```
SELECT {+ EXPLAIN} *
FROM customer c, cust_calls u
WHERE c.customer_num = u.customer_num

SELECT {+ EXPLAIN AVOID_EXECUTE} *
FROM customer c, cust_calls u
WHERE c.customer_num = u.customer_num
```



Agenda

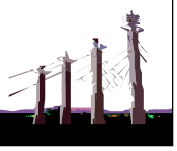
- Introducing optimizer directives
- Types of optimizer directives
- How to use optimizer directives
- External optimizer directives

Control of optimizer directives

- ONCONFIG parameter
  - DIRECTIVES 1/0 (1: on, 0: off, default on)
  - Affects entire instance
- environment variable
  - IFX_DIRECTIVES 1/0 or on/off
  - Affects all sessions starting from this environment
  - If set, override ONCONFIG setting

Syntax

- Optimizer directives are specified as “comments” in a query
- {+ directives }
- --+ directives
  till the end of line
- /*+ directives */



Syntax - example

```
SELECT {+ INDEX (customer)} lname, fname
FROM customer WHERE state = “CA”

SELECT --+ INDEX (customer)
lname, fname
FROM customer WHERE state = “CA”

SELECT /*+ INDEX (customer) */ lname, fname
FROM customer WHERE state = “CA”
```

Syntax – explain file

SELECT **{+ INDEX (customer)}** lname, fname
FROM customer WHERE state = "CA"

DIRECTIVES FOLLOWED:
INDEX (customer)
DIRECTIVES NOT FOLLOWED:

SELECT **--+ INDEX (customer)**
lname, fname
FROM customer WHERE state = "CA"

DIRECTIVES FOLLOWED:
INDEX (customer)
DIRECTIVES NOT FOLLOWED:

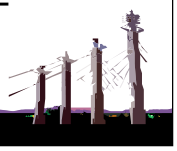
SELECT **/*+ INDEX (customer) */** lname, fname
FROM customer WHERE state = "CA"

DIRECTIVES FOLLOWED:
INDEX (customer)
DIRECTIVES NOT FOLLOWED:

Used in combination

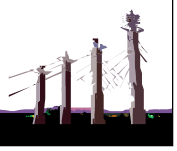
- Multiple directives can be specified in a query

```
SELECT
  {+ INDEX(c) USE_HASH(u) ORDERED}
  c.lname, c.fname, u.call_dtime, u.call_descr
FROM customer c, cust_calls u
WHERE c.customer_num = u.customer_num
```



Directives combination – explain file

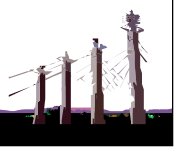
```
SELECT {+ INDEX(c) USE_HASH(u) ORDERED}
  c.lname, c.fname, u.call_dtime, u.call_descr
FROM customer c, cust_calls u  WHERE c.customer_num = u.customer_num

DIRECTIVES FOLLOWED:
INDEX( c )
USE_HASH( u )
ORDERED
DIRECTIVES NOT FOLLOWED:

Estimated Cost: 14
Estimated # of Rows Returned: 7

  1) usr1.c: INDEX PATH

    (1) Index Keys: customer_num   (Serial, fragments: ALL)

  2) usr1.u: SEQUENTIAL SCAN

DYNAMIC HASH JOIN
    Dynamic Hash Filters: usr1.c.customer_num = usr1.u.customer_num
```



Multiple directives - separator

- Both space and comma can be used to separate multiple directives

```
SELECT  {+ INDEX(c) USE_HASH(u) ORDERED}
  c.lname, c.fname, u.call_dtime, u.call_descr
FROM customer c, cust_calls u
WHERE c.customer_num = u.customer_num

SELECT  {+ INDEX(c), USE_HASH(u), ORDERED}
  c.lname, c.fname, u.call_dtime, u.call_descr
FROM customer c, cust_calls u
WHERE c.customer_num = u.customer_num
```



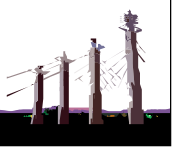
Directives not followed

- Not all directives specified can be followed
  - directives may have syntax error
  - directives may conflict with each other
  - directives may rule out all possible paths

Directives – syntax error

- Directives with syntax error are ignored

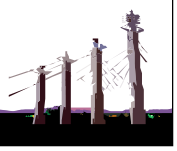
```
SELECT {+ AVOID_INDEX(c)} lname, fname
FROM customer c WHERE state = "CA"

DIRECTIVES FOLLOWED:
DIRECTIVES NOT FOLLOWED:
Estimated Cost: 2
Estimated # of Rows Returned: 25

 1) usr1.c: SEQUENTIAL SCAN

     Filters: usr1.c.state = 'CA'
```



Directives - conflict

- If two directives are conflicting with each other, both will be disallowed

```
SELECT {+ USE_NL(c) ORDERED} *
FROM customer c, cust_calls u
WHERE c.customer_num = u.customer_num

DIRECTIVES FOLLOWED:
DIRECTIVES NOT FOLLOWED:
USE_NL ( c ) Cannot use USE_NL for first table with ORDERED.
ORDERED  Cannot use USE_NL for first table with ORDERED.
```



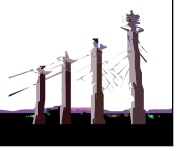
Directives – rule out all access path

- If combination of directives rule out all possible access path, then all relevant directives will be disallowed

```
SELECT
  {+ AVOID_FULL(c) AVOID_INDEX(c cidx1 cidx2)} *
FROM customer c where state = "CA"

DIRECTIVES FOLLOWED:
DIRECTIVES NOT FOLLOWED:
AVOID_FULL ( c ) Directives rule out all access paths for table.
AVOID_INDEX ( c cidx1 cidx2 ) Directives rule out all access
    paths for table.
```

Directives - rules

- Cannot specify multiple directives of the same type
  - multiple access method directives on the same table
    - only exception is AVOID_FULL, AVOID_INDEX
  - multiple join method directives on the same table
  - multiple optimization goals directives
- Cannot specify join method directives on all tables in a query
- Join order and join method directives cannot interfere with outer joins



Directives - views

- Directives can be used in query used to define a view
- Access method directives and join method directives cannot be used on a view in a query

Directives - subquery

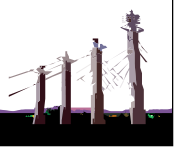
- Access method directives, join method directives and join order directive can be used in a subquery
- Optimization goal and explain directives cannot be used in a subquery
- Cannot have join order directives in both main query and subquery

Directives – ANSI JOIN

- Support for directives in ANSI JOIN query added in 11.10 (Cheetah) release
- Access method directives are supported
- Join method directives are not supported (only nested-loop join for ANSI JOIN) unless query can be transformed to informix join
- Join order directives supported but cannot really be enforced
- Optimization goal and explain directives supported

Agenda

- Introducing optimizer directives
- Types of optimizer directives
- How to use optimizer directives
- External optimizer directives



External directives

- Used for situations where query text cannot be modified directly (e.g., packaged application)
- Saved in sysdirectives catalog table
- Introduced in IDS 10
- Only user “informix” or DBA can create external optimizer directives

External directives - syntax

SAVE EXTERNAL DIRECTIVES directives
[ACTIVE | INACTIVE | TEST ONLY] FOR query

- directives are optimizer directives valid for the query.
- query is the text of a valid SELECT, UPDATE, DELETE statement.

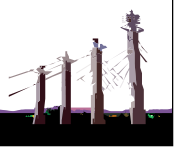
Inactive(0): Server ignores the saved external directives for the query.

Test only(1): Server applies the directives only to matching queries that DBA or user Informix executes.

Active(2): Server applies the directives to any subsequent query that matches the query string if external directive is enabled.

External directives - example

```
save external directives
{+ ORDERED USE_NL(u)} active for
SELECT * FROM customer c, cust_calls u
WHERE c.customer_num = u.customer_num

SELECT * FROM customer c, cust_calls u
WHERE c.customer_num = u.customer_num
```

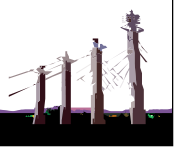


External directives – explain file

```
SELECT * FROM customer c, cust_calls u
WHERE c.customer_num = u.customer_num

External Directives in effect.
DIRECTIVES FOLLOWED:
ORDERED
USE_NL ( u )
DIRECTIVES NOT FOLLOWED:

Estimated Cost: 9
Estimated # of Rows Returned: 7

  1) usr1.c: SEQUENTIAL SCAN

  2) usr1.u: INDEX PATH

    (1) Index Keys: customer_num call_dtime   (Serial, fragments: ALL)
        Lower Index Filter: usr1.c.customer_num = usr1.u.customer_num
NESTED LOOP JOIN
```

The sysdirectives table

```
id              2
query
select * from customer c, cust_calls u
  WHERE c.customer_num = u.customer_num
directive
 ORDERED USE_NL(u)
directivecode   <BYTE value>
active          1
hashcode        1813896241
```

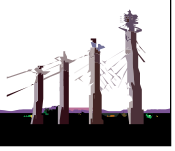


External directives - manipulation

- Only user “informix” can update/delete entries in sysdirectives table
- Disable the entry

  UPDATE sysdirectives set active = 0 where id = 2
- Remove the entry

  DELETE from sysdirectives where id = 2



Control of external directives

- ONCONFIG parameter
  EXT_DIRECTIVES 0/1/2 (default is 0 – off)
- Environment variable
  IFX_EXTDIRECTIVES 0/1

	EXT_DIRECTIVES (0)	EXT_DIRECTIVES (1)	EXT_DIRECTIVES (2)
IFX_EXTDIRECTIVES (Not Set)	OFF	OFF	ON
IFX_EXTDIRECTIVES (0)	OFF	OFF	OFF
IFX_EXTDIRECTIVES (1)	OFF	ON	ON

External directives – Misc.

- Query matching: must be exact match (including new line, white space, comments), only allow white space mismatches at beginning and end of query string
- Query string can contain host variables

  save external directives {+ INDEX (c) } active for

  SELECT * FROM customer c WHERE state = ?

Summary

- Optimizer directives can be used to direct query optimizer to select desired query plan
- Effect of optimizer directives can be monitored in sqexplain.out file
- External optimizer directives can be used to associate a set of directives with a specific query, without embedding the directives in the query itself

Session A16 (83)

Improving SQL Query Performance - Optimizer Directives

Bingjie Miao
IBM
bingjie@us.ibm.com