IBM Software Group

Advanced DB2/LUW Statistics with Statistical Views

John Hornibrook

Manager

DB2 Query Optimization Development

DB2 Information Management Software

ON DEMAND BUSINESS™

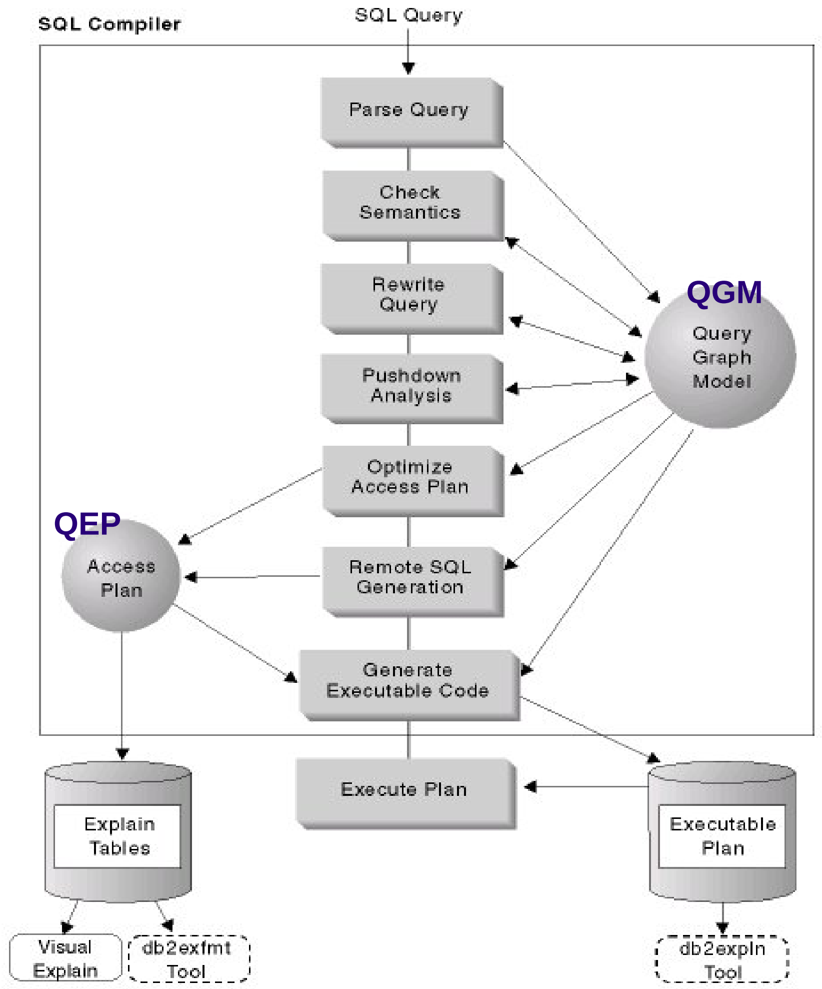
Objectives

- Discuss powerful new form of DB statistic
  - Helps the query optimizer choose better Query Execution Plans (QEPs)
  - Solves many of the costing limitations with current statistics
- Provide skills in determining when and how to use statistical views
- Deeper understanding of query optimization
- Skills in interpreting QEPs via the Explain facility

Agenda

- Query optimization overview
  - DB Statistics
  - Cardinality estimation
  - Cost estimation
- Statistical views
  - Concepts
  - Usage scenarios
  - Best practices
  - Technical reference
    - ALTER VIEW
    - RUNSTATS

Query Optimization Overview



Phases of SQL Compilation

Parsing

- Catch syntax errors
- Generate internal representation of query

Semantic checking

- Determine if query makes sense
- Incorporate view definitions
- Add logic for constraint checking and triggers

Query optimization

- **Modify query to improve performance (Query Rewrite)**
- **Choose the most efficient "access plan" (Query Optimization)**

Threaded code generation

- Generate efficient "executable" code

Query Optimization Overview

- **Given** a non-procedural query, the schema, the hardware and data server configuration, data and physical layout statistics
- **Determine** the optimal specification of the query and the optimal query execution plan
- **By**
  - **rewriting** the query into a canonical form
  - **generating** alternative query execution plans
  - **modeling** IO, CPU, memory, communication usage of each alternative
  - **selecting** a minimal cost alternative for execution
- **Because we have** INCREASING database sizes, database design complexity, query complexity, demand for quick response and system complexity

Query Optimization Overview

- **The Optimizer generates alternative**
  - Operation orders
  - Implementations to use
  - Location strategies
- **The Optimizer Cost Model**
  - Estimates Cardinality
    - Biggest impact on estimated cost!
  - Estimates CPU, I/O, Communication and memory costs
  - Considers I/O parallelism (prefetch), CPU parallelism (SMP & EEE)
- **Plan Evaluation and Search Strategies**
  - Plans using a bottom up approach
  - Selects the best plan
  - Considers Dynamic v/s Greedy, Bushy v/s Deep plans
  - Prunes plans and sub-plans when appropriate

Optimizer Influences

- **Database layout**
  - Schema, including indexes and constraints
  - Table partitioning in a partitioned database
- **Statistics**
  - Statistics on tables, columns, indexes, etc. collected by RUNSTATS
- **Configuration parameters, e.g.**
  - Speed of CPU
  - Storage device characteristics
  - Communications bandwidth
  - Degree of parallelization
- **Optimization Level**
- **Memory resources**
  - Buffer pool(s)
  - Sort heap
- **Concurrency Environment**
  - Average number of users
  - Isolation level / blocking / number of available locks

Catalog Statistics Used by the Optimizer

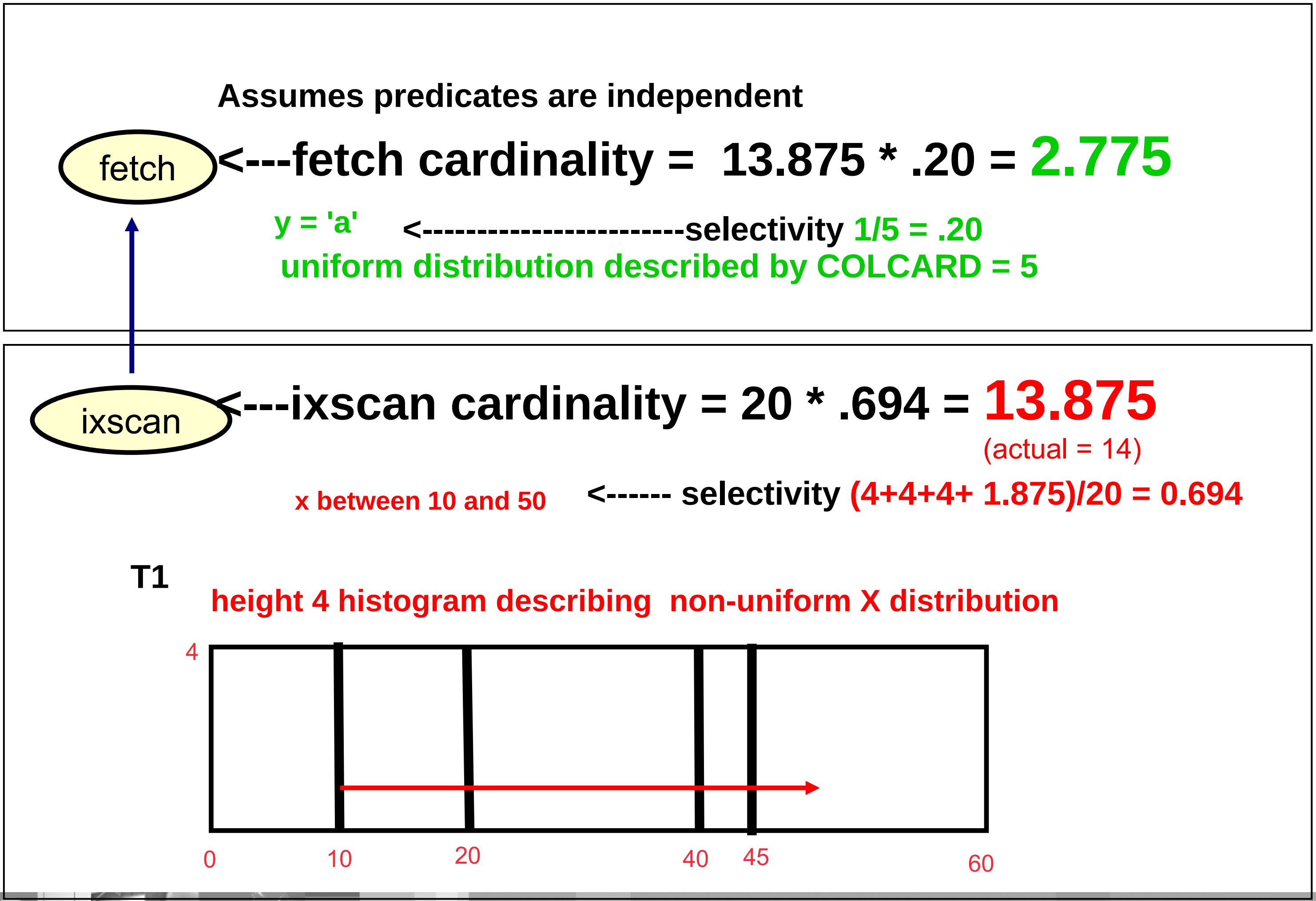
- **Basic Statistics**
  - ► **no. of rows/pages/active blocks in table**
  - ► **for each column in a table, optionally records**
    - –**no. of data values, avg. length of data values, data range information**
- **Non-uniform distribution statistics**
  - ► **N most frequent values (default 10)**
    - –**good for <u>equality</u> predicates**
  - ► **M quantiles (default 20)**
    - –**good for <u>range</u> predicates**
  - ► **N and M set by DBA as DB configuration parameters or specified via RUNSTATS utility**
- **Multi-column statistics**
  - **Column group statistics**
    - **No. of distinct values for a group of columns**
- **Index clustering (DETAILED index statistics)**
  - ► **used to better estimate data page fetches**
  - ► **empirical model: determines curve of I/O vs. buffer size**
  - ► **accounts for benefit of large buffers**

Cardinality Estimation – Local predicates

T1

x	y
0	a
1	b
3	c
9	d
12	e
19	b
19	d
20	d
30	e
31	a
32	c
39	d
42	e
43	a
44	b
45	d
47	e
50	a
55	b
60	c

SELECT * FROM T1 WHERE x BETWEEN 10 AND 50 AND y = 'a'



Assumes predicates are independent

<---fetch cardinality = 13.875 * .20 = 2.775

y = 'a' **<-------------------------selectivity 1/5 = .20**

uniform distribution described by COLCARD = 5

<---ixscan cardinality = 20 * .694 = 13.875

(actual = 14)

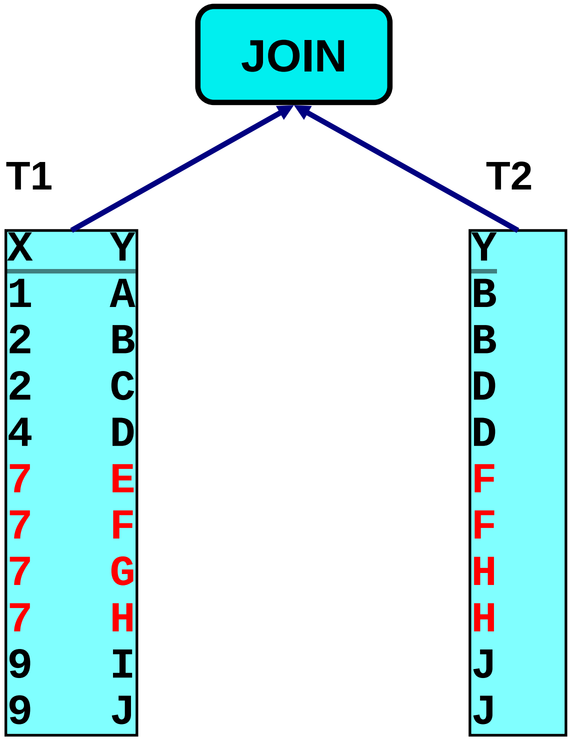
x between 10 and 50 **<------ selectivity (4+4+4+ 1.875)/20 = 0.694**

T1

height 4 histogram describing non-uniform X distribution

Cardinality Estimation – Local and join predicates

SELECT * FROM T1, T2 WHERE T1.x = 7 AND T1.y = T2.y



T1:

X	Y
1	A
2	B
2	C
4	D
7	E
7	F
7	G
7	H
9	I
9	J

T2:

Y
B
B
D
D
F
F
H
H
J
J

SYSSTAT.COLDIST (X)

TYPE	SEQNO	COLVALUE	VALCOUNT
F	1	7	4
F	2	9	2
F	3	2	2

Selectivity (T1.x = 7):
= 4/10
Using frequent value statistics

Selectivity (T1.y = T2.y):
= 1 / max(colcard(T1.y), colcard(T2.y))
= 1 / max(10,5)
= 1/10

Join predicate selectivity assumes:

- **Inclusion:**
  - **All values in T2.y are included in domain of T1.y**
- **Uniformity:**
  - **Values are uniformly distributed in both columns**

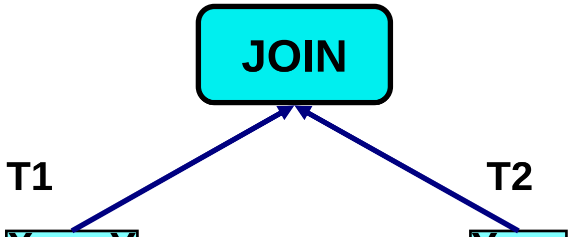
Result cardinality:
= Card(T1) * Card(T2) * sel(T1.x=7) * sel(T1.y=T2.y)
= 10 * 10 * 0.4 * 0.1
= 4

Actual: 4

Cardinality Estimation – Local and join predicates

What if the join predicate assumptions are incorrect?

SELECT * FROM T1, T2 WHERE T1.x = 1 AND T1.y = T2.y



T1

X	Y
1	A
2	B
2	C
4	D
7	E
7	F
7	G
7	H
9	I
9	J

T2

Y
A
A
A
A
A
F
M
N
N
N

SYSSTAT.COLDIST (X)

TYPE	SEQNO	COLVALUE	VALCOUNT
F	1	7	4
F	2	9	2
F	3	2	2

Selectivity (T1.x = 1):
= 1/10
Using frequent value statistics

Selectivity (T1.y = T2.y):
= 1 / max(colcard(T1.y), colcard(T2.y))
= 1 / max(10,5)
= 1/10

Result cardinality:
= Card(T1) * Card(T2) * sel(T1.x=1) * sel(T1.y=T2.y)
= 10 * 10 * 0.1 * 0.1
= 1

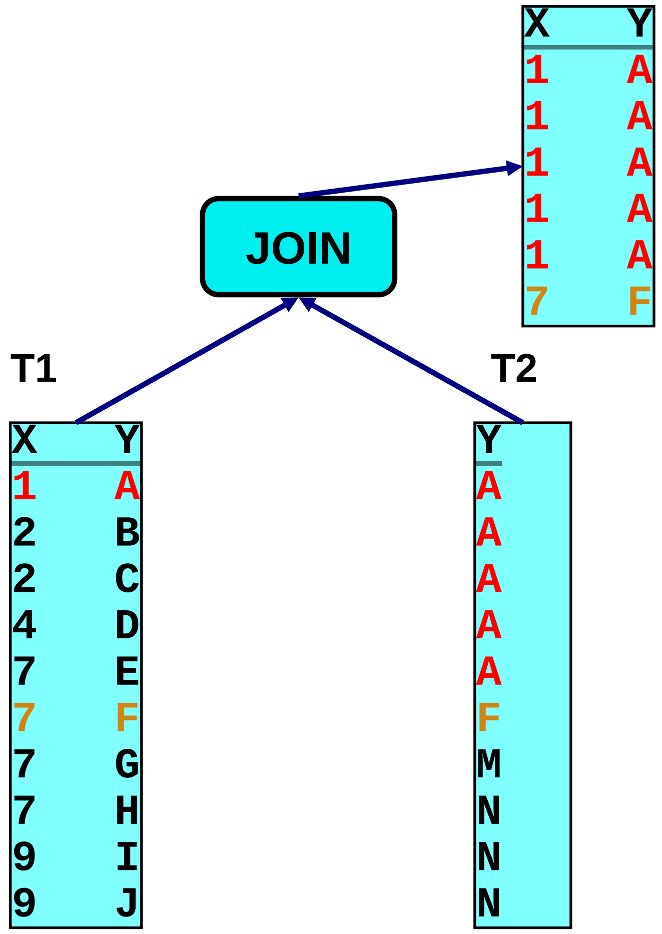
Actual: 5

Error: (5-1)/5 = 80% !!

Cardinality Estimation – Local and join predicates

Consider the data distributions after the join

SELECT * FROM T1, T2 WHERE T1.y = T2.y



X	Y
1	A
1	A
1	A
1	A
1	A
7	F

T1

X	Y
1	A
2	B
2	C
4	D
7	E
7	F
7	G
7	H
9	I
9	J

T2

Y
A
A
A
A
A
F
M
N
N
N

SYSSTAT.COLDIST (X)

TYPE	SEQNO	COLVALUE	VALCOUNT
F	1	1	5
F	2	7	1

Compute local predicate selectivity based on join result distribution:

Selectivity (T1.x = 1) AND (T1.y = T2.y)
= 5/6
Result cardinality:
= Card(T1 join T2) * sel(T1.x = 1)
= 6 * 5/6
= 5
Selectivity (T1.x = 7) AND (T1.y = T2.y)
= 1/6
Result cardinality:
= 6 * 1/6
= 1

Statistical Views

Create a view to represent the join:

```
CREATE VIEW SV1 (T1_x) AS
 (SELECT T1.x FROM T1, T2 WHERE T1.y = T2.y)
```

Associate statistics with the view:

SYSSTAT.TABLES

TABNAME	CARD
SV1	6

SYSSTAT.COLUMNS

TABNAME	COLNAME	COLCARD
SV1	T1_X	2

SYSSTAT.COLDIST (SV1.T1_X)

TYPE	SEQNO	COLVALUE	VALCOUNT
F	1	1	5
F	2	7	1

Match queries to the view:

```
SELECT * FROM T1, T2 WHERE T1.x = 1 AND T1.y = T2.y ✓
SELECT * FROM T1, T2 WHERE T1.x = 7 AND T1.y = T2.y ✓
SELECT * FROM T1, T2 WHERE T1.y = T2.y ✓
SELECT * FROM T1, T2 WHERE T1.x = 1 ✗
```

Must include view's predicates

Use view's statistics to improve cardinality estimation!

Statistical Views

- A powerful way to represent data statistics for query specifications
  - complex predicates

    ```
    WHERE NAME LIKE ‘%JONES%’
    ```
  - relationships among complex predicates

    ```
    WHERE PRODUCT = ‘DVD’ AND PRICE < ’15.00’
    ```
  - relationships across tables

    ```
    SELECT … FROM CUSTOMER C, FACT F
    WHERE C.NAME = ‘Popular Customer’ AND
              C.CUST_ID = F.CUST_ID
    ```

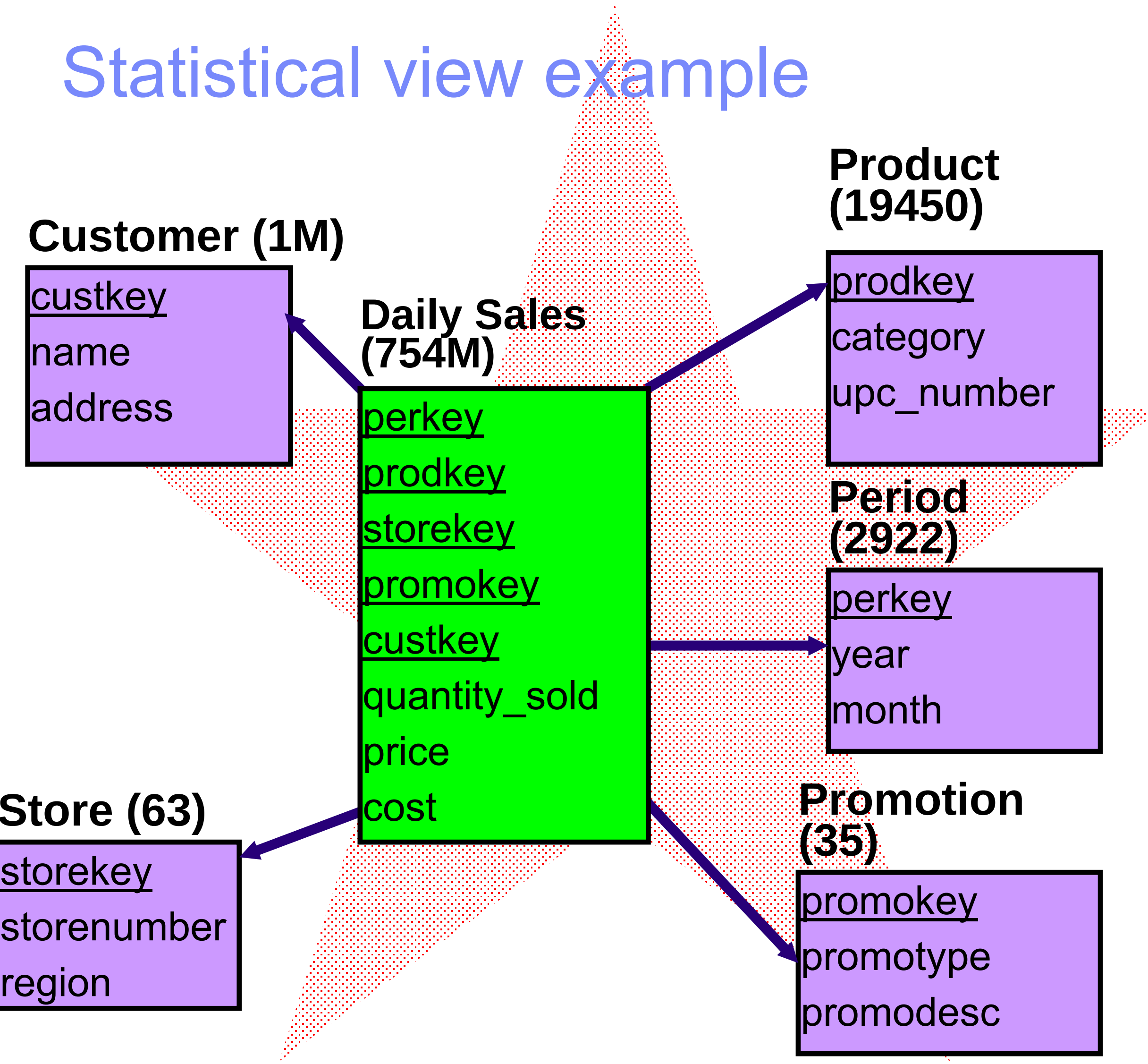
Statistical Views

- Statistics are associated with the view
- Query does <u>not</u> need to reference the view
- Materialized Query Table (MQT) matching technology matches query to statistical view
- View is NOT materialized
- Statistics on the view are used to ‘adjust’ selectivity estimates for predicates in query

Enabling a view for optimization

- ALTER VIEW statement extended
  - Enable a view for optimization:
    - ALTER VIEW DB2USER.STAT_VIEW1 ENABLE QUERY OPTIMIZATION
  - Disable a view for optimization
    - ALTER VIEW DB2USER.STAT_VIEW1 DISABLE QUERY OPTIMIZATION
- RUNSTATS command extended
  - RUNSTATS ON TABLE DB2USER.STAT_VIEW1 WITH DISTRIBUTION

Statistical view example



- Consider a ‘star’ schema
- Logical DB design resembles a star
- Central table contains business ‘facts’
  - Sales prices, cost, quantities, etc.
- Surrounding tables contain ‘dimensional’ data
  - Time, location, characteristics, etc.
- Each dimension is a ‘parent’ of the fact table
  - 1:N from a dimension to the fact

Statistical view example

- Determine whether consumers will buy a product again if they are offered a discount on a return visit (‘returned customers’)
- The study is only done for store ‘01’, which has 18 locations nationwide.
- Query:

```
SELECT  count(*)
FROM  store s, promotion p, daily_sales f
WHERE  s.storekey = f.storekey
    AND  p.promokey = f.promokey
    AND  s.store_number = ’01’
    AND  p.promotype = 1
```

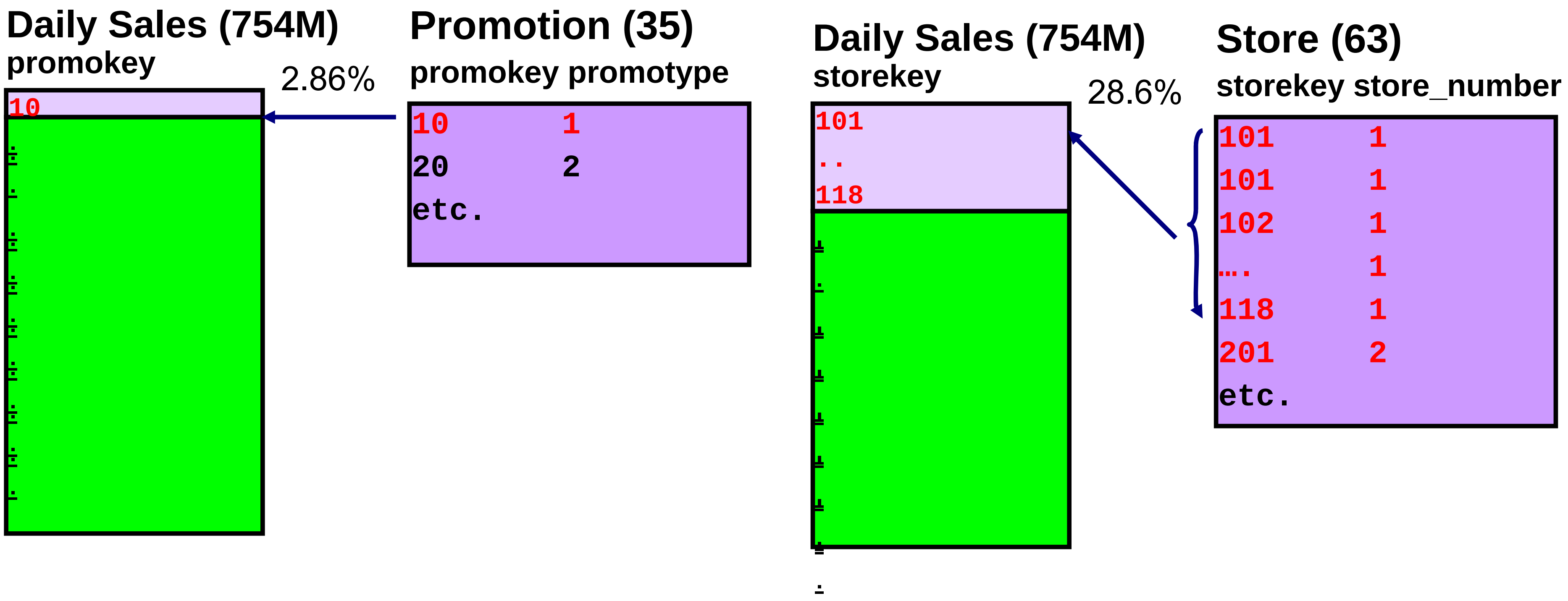
Statistical view example

- Distribution of promotion types in PROMOTION dimension
- ‘Returned customer’ is 1/35 (2.86%) of all promotion types

PROMOTION *(35 rows)*				
PROMOTYPE	**PROMODESC**	**COUNT(promotype)**		**Percentage promotions**
1	**Returned customers**	**1**		**2.86%**
2	Coupon	15		42.86%
3	Advertisement	5		14.29%
4	Manager’s special	3		8.57%
5	Overstocked items	4		11.43%
6	End aisle display	7		20.00%

Statistical view example

- Store '01' represents 18 of 63 stores in the store dimension (28.6%)
- 'Returned customer' represents 1 of 35 promotions in the promotion dimension (2.86%)
- Assume store key and promotion key:
  - occur uniformly within the fact table
  - are independent



Statistical view example

Result cardinality computation (simplified):

= Card(DAILY_SALES) * 1/35 * 18/63

= 7.54069e+08 * 0.0286 * 0.286

= 6.15567e+06

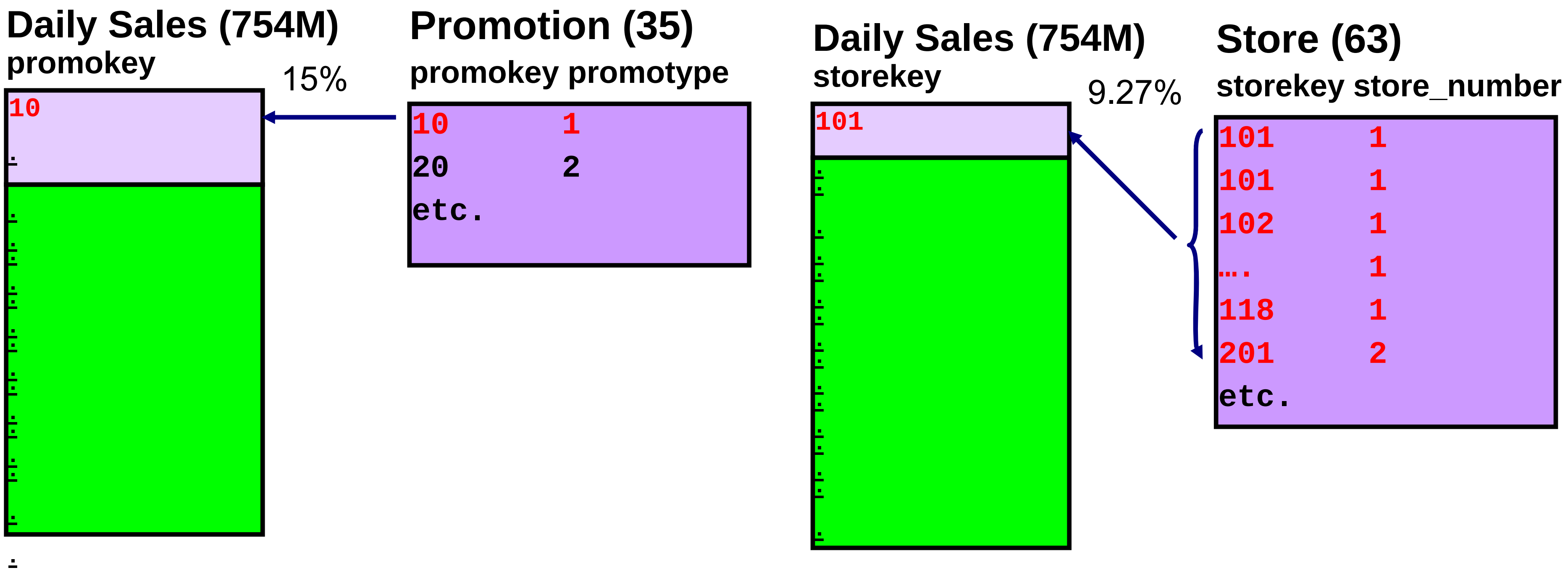
Star join index ANDing QEP

- **Start with PROMOTION because it appears the most filtering**

```
                                              6.15567e+06
                                                IXAND
                                                (   8)
                                /------------------+------------------\
  1/35 * 7.54069e+08 →   2.15448e+07         18/63 * 7.54069e+08 →  2.15448e+08
                           NLJOIN                                     NLJOIN
                           (   9)                                     (  13)
                    /---------+--------\                       /---------+--------\
                  1                  2.15448e+07            18                1.19694e+07
               FETCH                   IXSCAN            FETCH                   IXSCAN
               (  10)                  (  12)            (  14)                  (  16)
              /---+---\                  |              /---+---\                   |
            35         35           7.54069e+08        18         63          7.54069e+08
         IXSCAN   TABLE: DB2DBA    INDEX: DB2DBA    IXSCAN   TABLE: DB2DBA  INDEX: DB2DBA
         (  11)   PROMOTION        PROMO_FK_IDX     (  15)   STORE           STORE_FK_IDX
           |                                           |
           35                                         63
     INDEX: DB2DBA                                 INDEX: DB2DBA
    PROMOTION_PK_IDX                              STOREX1
```

Statistical view example

- What if uniformity assumption is incorrect?
- Consider a different fact table distribution (closer to reality!)
  - ‘Returned customers’ are 15% of fact table
  - Store 1 is 9.27% of fact table



Statistical view example

Create a statistical view for (store-daily_sales) and (promotion-daily_sales) joins

```
CREATE VIEW DB2DBA.SV_STORE AS
 (SELECT S.*
 FROM STORE S, DAILY_SALES F
 WHERE S.STOREKEY = F.STOREKEY)

CREATE VIEW DB2DBA.SV_PROMO AS
 (SELECT P.*
 FROM PROMOTION P, DAILY_SALES F
 WHERE S.PROMOKEY = F.PROMOKEY)
```

Include all dimension columns
Don’t need to include fact table columns

Statistical view example

Enable statistical views for query optimization

```
ALTER VIEW DB2DBA.SV_STORE ENABLE QUERY OPTIMIZATION
ALTER VIEW DB2DBA.SV_PROMO ENABLE QUERY OPTIMIZATION
```

Gather statistics for the statistical views:

```
RUNSTATS ON TABLE DB2DBA.SV_STORE WITH DISTRIBUTION
RUNSTATS ON TABLE DB2DBA.SV_PROMO WITH DISTRIBUTION
```

Statistical view example

Result cardinality computation (simplified):

= Card(DAILY_SALES) * 0.0927 * 0.1496

= 7.54069e+08 * 0.0927 * 0.1496

= 1.04627e+07

Star join index ANDing QEP

- **Start with STORE because it appears the most filtering**

From SV_STORE → 6.99152e+07

From SV_PROMO → 1.12845e+08

```
                                         1.04627e+07
                                         IXAND
                                         (   8)
                         /------------------+------------------\
                  6.99152e+07                                1.12845e+08
                    NLJOIN                                     NLJOIN
                    (   9)                                     (  13)
             /---------+--------\                       /---------+--------\
          18                 3.88418e+06             1                  1.12845e+08
        FETCH                   IXSCAN             FETCH                   IXSCAN
        (  10)                  (  12)             (  14)                  (  16)
       /---+---\                   |              /---+---\                   |
    18           63           7.54069e+08      35           35          7.54069e+08
  IXSCAN     TABLE: DB2DBA    INDEX: DB2DBA   IXSCAN    TABLE: DB2DBA  INDEX: DB2DBA
  (  11)     STORE            STORE_FK_IDX     (  15)    PROMOTION     PROMO_FK_IDX
    |                                           |
    63                                         35
INDEX: DB2DBA                                INDEX: DB2DBA
STOREX1                                    PROMOTION_PK_IDX
```

Statistics (DB2 9)

- RUNSTATS gathers statistics by executing a query against the view
  - Gathers statistics on the query result
  - Can specify columns or column groups
- Only 'data' statistics are gathered:
  - Cardinality, column cardinality, data distributions, etc
- Not physical layout statistics:
  - Data pages, file pages, active blocks, clustering, etc.
  - Because the statistical view is not materialized on disk
  - Consequently, no indexes can be created
- Unsupported RUNSTATS options:
  - UTIL_IMPACT_PRIORITY (throttling)
  - TABLESAMPLE SYSTEM
  - Any Indexes Clause options
  - Any ON KEY COLUMNS option
- Row-level sampling is supported

Statistics

- UPDATE statistics is supported
  - An alternative to RUNSTATS is to run sampling queries and ‘manually’ update the statistics
  - MUST update statistics manually in V8.2
    - RUNSTATS on statistical view is supported, but only to insert empty rows into system catalog tables
  - A tool is available upon request
- Best practices
  - The statistical view MUST have statistics
    - That is the whole point ☺
  - Always gather distribution statistics (frequent values and quantiles)
  - Statistical view statistics should be as good or better than the base table
    - i.e. if base table has distribution statistics, so too should the statistical view
    - At least same number of frequent values and quantiles (NUM_FREQVALS, NUM_QUANTILES options)
  - Consider LIKE statistics for queries with LIKE predicates

Considerations and limitations

- **Considerations:**
  - Statistical view cannot contain:
    - Aggregation or distinct operations
    - UNION, EXCEPT, or INTERSECT operations
    - Scalar aggregate (OLAP) functions
    - Warning returned during ALTER VIEW
      - **SQL20278W** The view "<viewname>" may not be used to optimize the processing of queries
    - Future releases will support more complex statistical views
  - Statistical views have many of the same matching restrictions as MQTs.
  - Use Explain diagnostic facility to understand why a statistical view or MQT may not have been used

Considerations and limitations

- **Minimize the number of statistical views**
  - Number of statistical views can impact compilation time
    - Query matching can be expensive
  - Best practice is to make the statistical view as general as possible
    - Don’t include local predicates
  - Typical usage
    - Create a statistical view for each dimension-fact join in a star schema
    - Limit to dimensions:
      - With skew in the fact table
      - Many more dimension ids than exist in fact table

Considerations and limitations

- If the statistical view becomes inoperative, it is no longer a statistical view
  - For example, a dependent table is dropped
  - Statistics are deleted from the catalogs
  - View must be recreated, re-enabled for optimization and RUNSTATS performed

Problem Determination (DB2 9)

- Explain diagnostic facility indicates which statistical views were considered
  - Doesn’t necessarily mean they changed the cardinality
- 2 new Explain tables in EXPLAIN.DDL
  - EXPLAIN_DIAGNOSTICS and EXPLAIN_DIAGNOSTICS_DATA
- Formatted message text appears in the db2exfmt output:

```
Extended Diagnostic Information:
--------------------------------
Diagnostic Identifier:    3
Diagnostic Details:       EXP0147W  The following statistical views may
  have been used by the optimizer to estimate cardinalities:
  “DB2DBA".“SV_STORE".
```

Problem Determination

- Extensive diagnostics about why an MQT or statistical view could not be used. Examples:

```
EXP0073W The following MQT or statistical view was not eligible
  because one or more data filtering predicates from the query could
  not be matched with the MQT: “DB2DBA".“SV_BAD“

EXP0066W The following MQT or statistical view was not eligible
  because an outer join or a subquery from the MQT or the query did
  not match: “DB2DBA".“SV_BAD“
```

Availability

- DB2 9
- Statistical views were available in V8.2 FP9 but weren't generally announced
  - Requires setting DB2_STATVIEW registry variable
  - No Runstats support
    - A tool is available upon request
  - Optimizer limitations
    - Create 2 table statistical views only
    - Must include all columns referenced by predicates in the view definition
- Material in this presentation is specific to DB2 9, except where otherwise noted.
- V8.2 documentation is available upon request

Future Enhancements

- Determining the necessary statistical views can be difficult
  - Statistical view advisor in a future release
- Large number of statistical views can degrade compilation time
  - > 50 is probably too many
  - Future improvements to filter inapplicable statistical views early
  - Future improvements to provide more general statistical view to cover more queries
- Runstats performance
  - Improved sampling support for certain types of joins
  - Push sampling down closer to the data access
- Auto-Runstats support

Summary

- Cardinality estimation is crucial to achieving optimal query execution plans
- Statistical views are a powerful new type of statistic to allow better cardinality estimation
- Particularly useful in star schema environments (Data Marts)
- Can be used for any schema

ALTER VIEW technical reference

- **The ALTER VIEW statement modifies an existing view by altering a reference type column to add a scope. It is now also used to toggle a regular view to a statistical view, and vice versa.**
  - **Authorization**
    - If ENABLE QUERY OPTIMIZATION or DISABLE QUERY OPTIMIZATION clause is used, the privileges held by the authorization ID of the statement must also include at least one of the following:
      - ALTER privilege on the *table* to be altered
      - CONTROL privilege on the *table* to be altered
      - SYSADM or DBADM authority
    - *"Tables"* means the tables or underlying tables of views referenced in the FROM clause of the view fullselect. Once again, all existing authorization rules applicable to the ALTER VIEW statement as outlined in the *DB2 SQL Reference* should still be satisfied.

ALTER VIEW technical reference

- **Syntax**

```
>>--ALTER VIEW--view name------------------------------------>
    .-----------------------------------------------------------------.
    v        .-COLUMN-.                                               |
>-+---ALTER--+--------+--column name--ADD SCOPE--+-typed table name-+-+-+-><
  |                                              '-typed view name--'   |
  '--+---ENABLE---+---QUERY OPTIMIZATION--------------------------------'
     '---DISABLE--'
```

- **Description**
  - ENABLE QUERY OPTIMIZATION
    - Indicates that the view's statistics can be used to improve the optimization of queries that overlap with the defining query of the view.
  - DISABLE QUERY OPTIMIZATION
    - Indicates that the view and any associated statistics will *not* be used to improve the optimization of queries that overlap with the defining query of the view. This is the default setting when a view is created.

ALTER VIEW technical reference

- **Notes**
  - A view cannot be enabled for optimization if:
    - The view directly or indirectly references an MQT. *(Note that an MQT or statistical view can reference a statistical view.)*
    - It is an inoperative view.
    - It is a typed view.
    - There is another view alteration in the same ALTER VIEW statement.
    - The user does not have alter privilege on both the view and the tables on which the view is defined.
  - If the definition of a view that is altered to enable optimization meets any of the conditions below, the ALTER VIEW ENABLE QUERY OPTIMIZATION will succeed (and a warning issued), but the optimizer will not exploit its statistics:
    - It contains aggregation or distinct operations.
    - It contains union, except, or intersect operations.
    - It contains scalar aggregate (OLAP) functions.