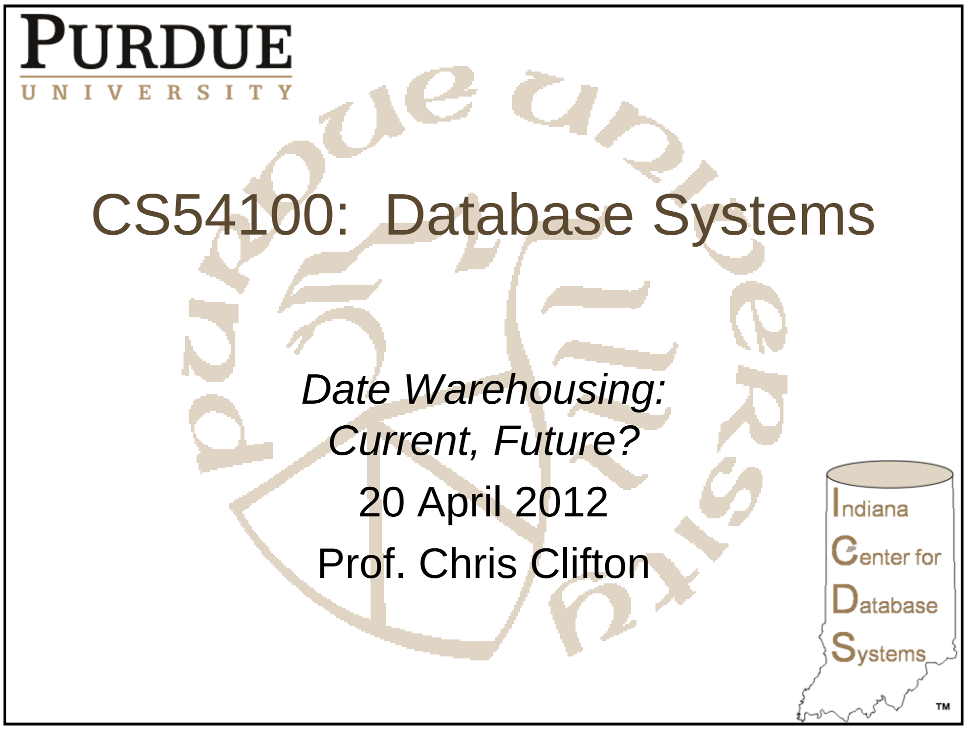


PURDUE
UNIVERSITY

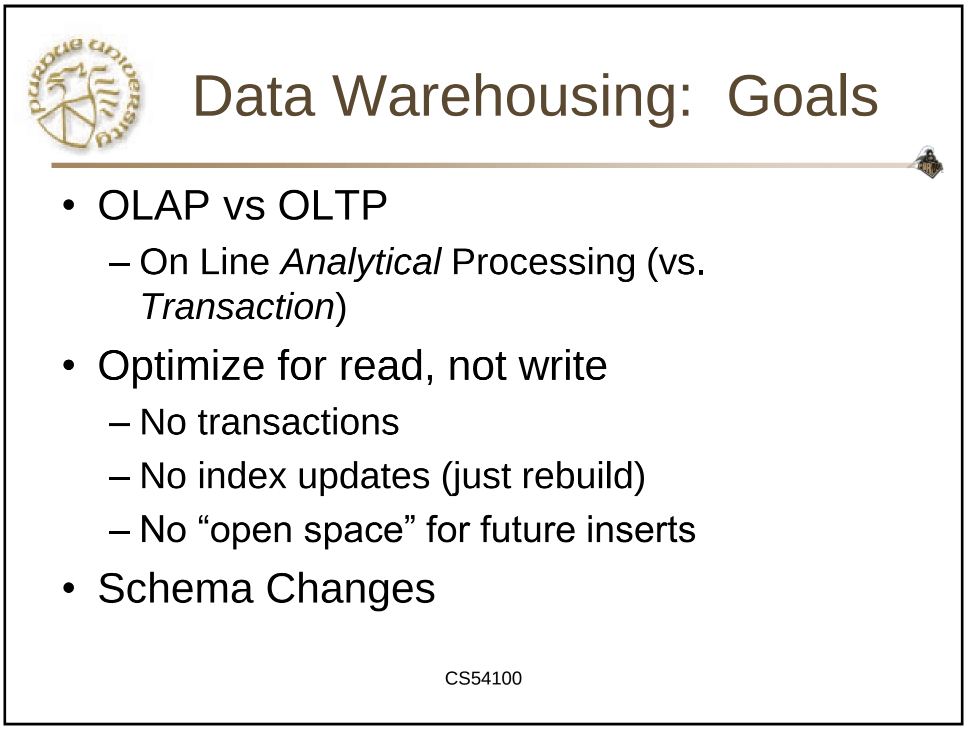
CS54100: Database Systems

Date Warehousing: Current, Future?

20 April 2012

Prof. Chris Clifton

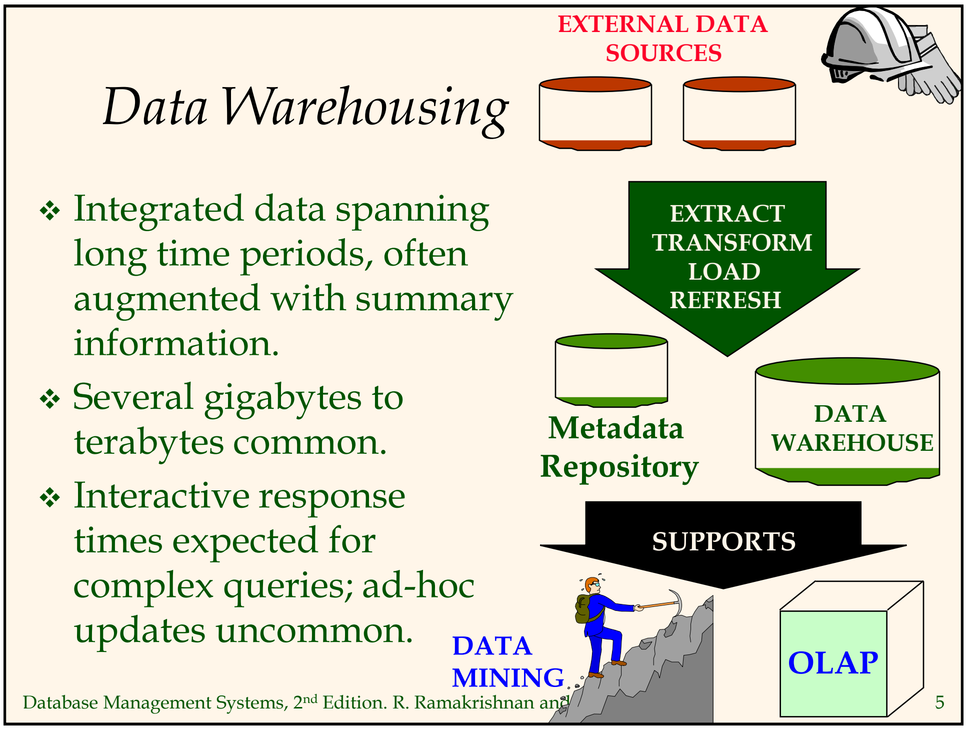
Indiana Center for Database Systems™



Data Warehousing: Goals

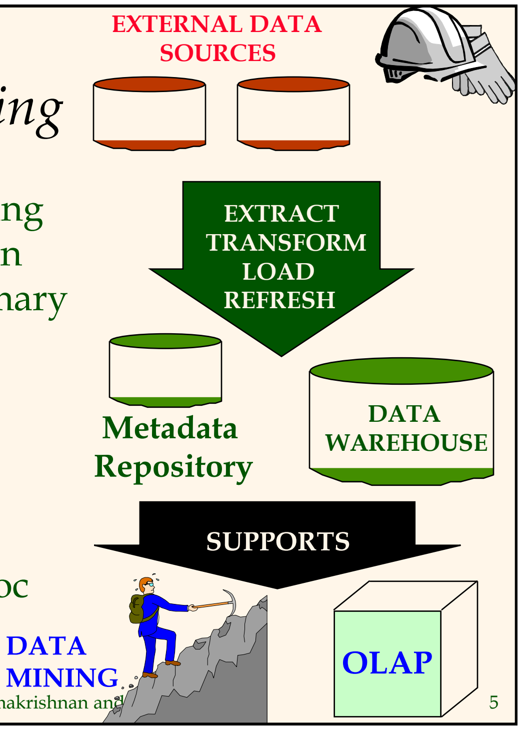
- OLAP vs OLTP
  - On Line *Analytical* Processing (vs. *Transaction*)
- Optimize for read, not write
  - No transactions
  - No index updates (just rebuild)
  - No “open space” for future inserts
- Schema Changes

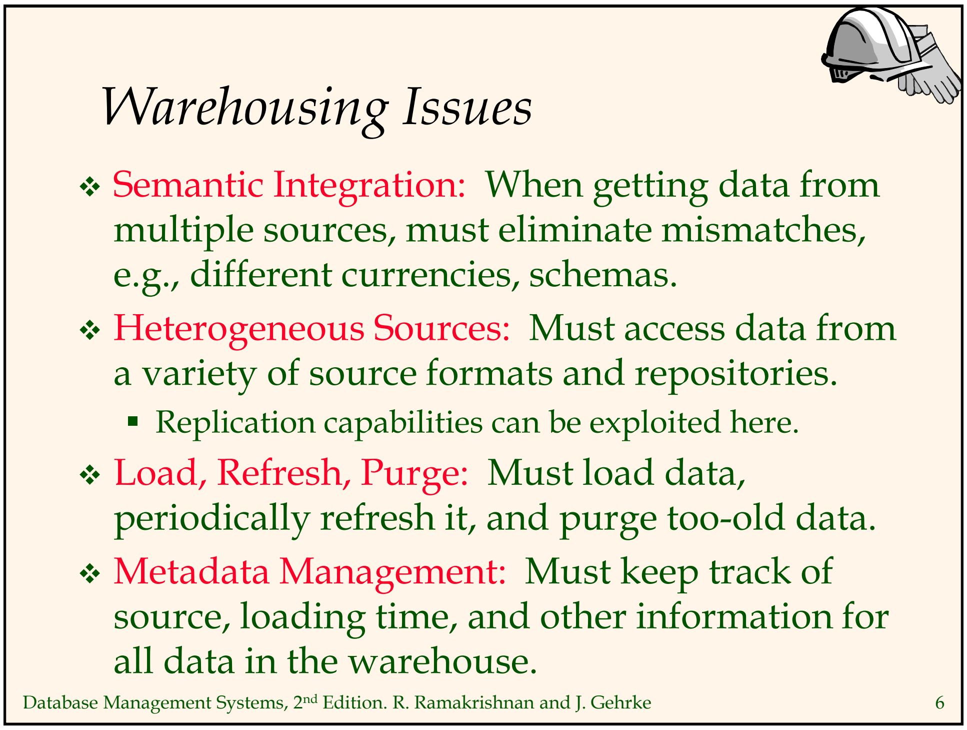
CS54100



Data Warehousing

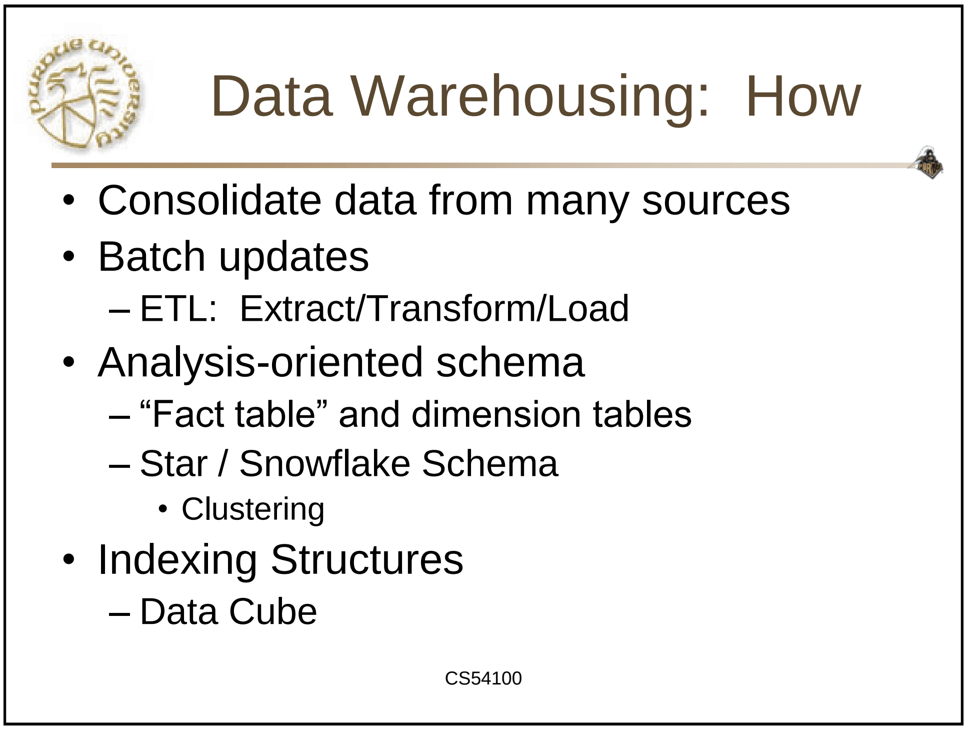
- Integrated data spanning long time periods, often augmented with summary information.
- Several gigabytes to terabytes common.
- Interactive response times expected for complex queries; ad-hoc updates uncommon.





Warehousing Issues

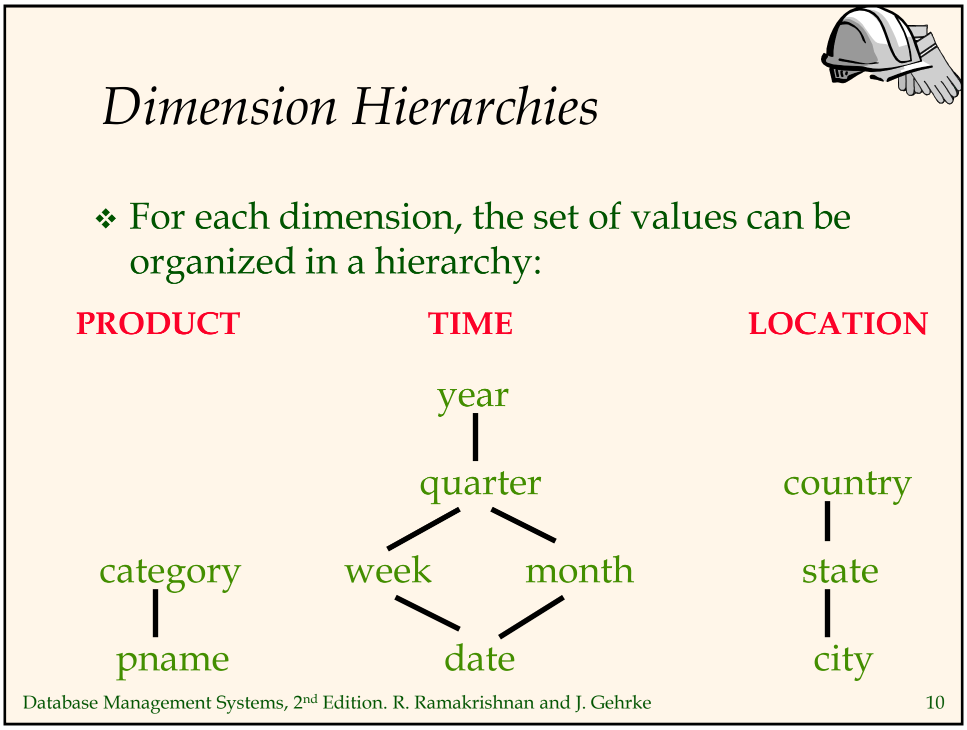
- Semantic Integration: When getting data from multiple sources, must eliminate mismatches, e.g., different currencies, schemas.
- Heterogeneous Sources: Must access data from a variety of source formats and repositories.
  - Replication capabilities can be exploited here.
- Load, Refresh, Purge: Must load data, periodically refresh it, and purge too-old data.
- Metadata Management: Must keep track of source, loading time, and other information for all data in the warehouse.



Data Warehousing: How

- Consolidate data from many sources
- Batch updates
  - ETL: Extract/Transform/Load
- Analysis-oriented schema
  - "Fact table" and dimension tables
  - Star / Snowflake Schema
    - Clustering
- Indexing Structures
  - Data Cube

CS54100

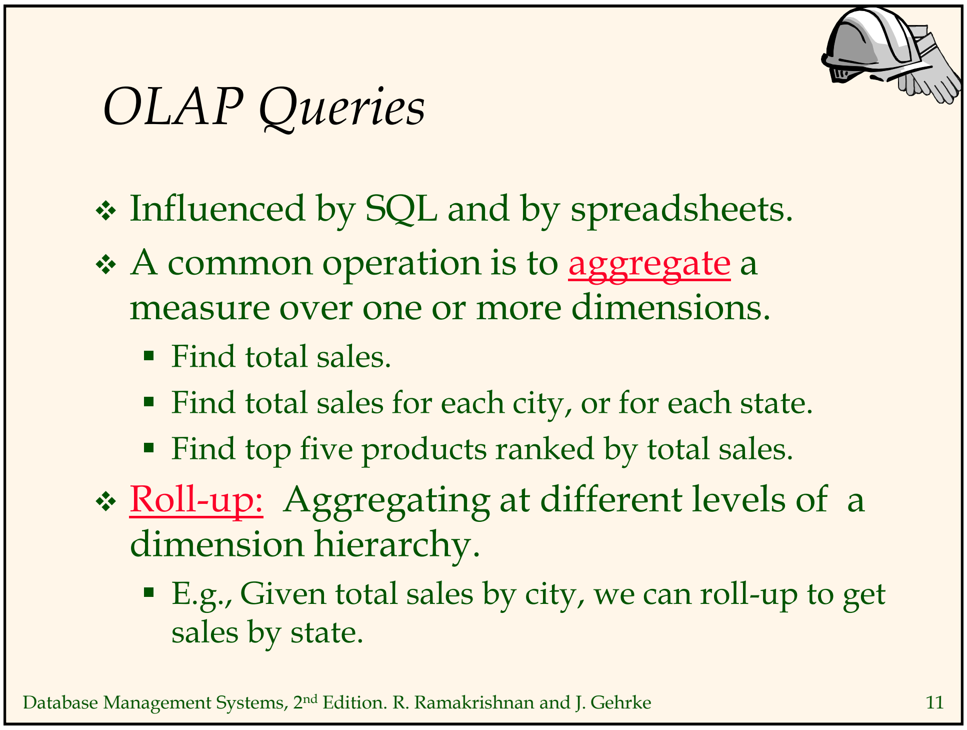


Dimension Hierarchies

- For each dimension, the set of values can be organized in a hierarchy:

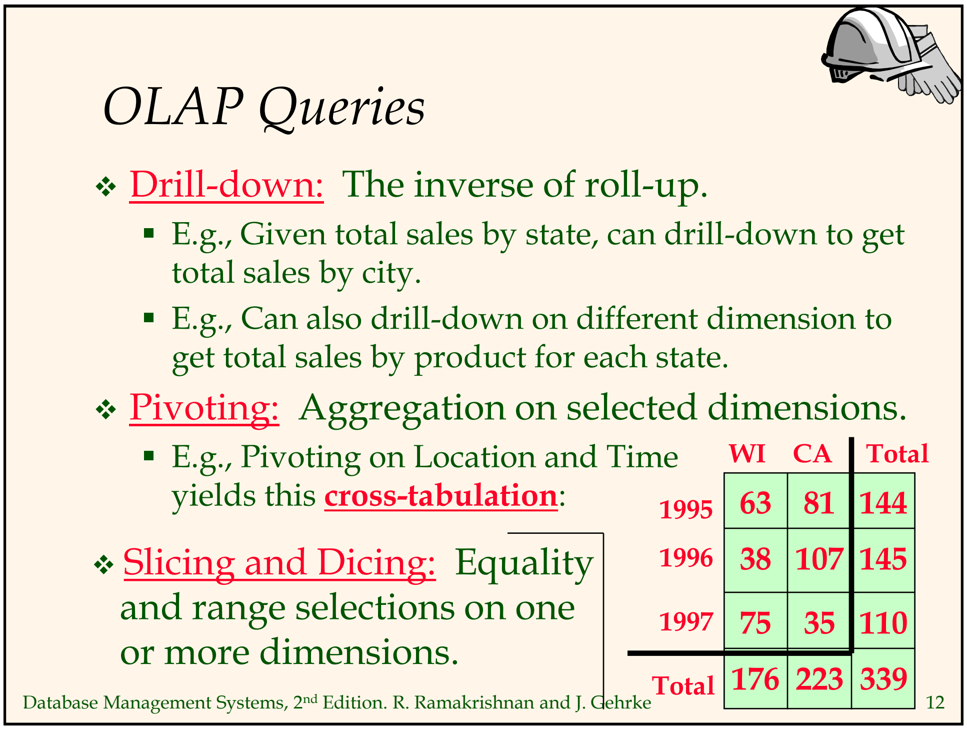
$$\begin{tikzcd}
\text{PRODUCT} & & \text{TIME} & & \text{LOCATION} \\
& & \text{year} \arrow[d, no head] & & \\
& & \text{quarter} \arrow[dl, no head] \arrow[dr, no head] & & \text{country} \arrow[d, no head] \\
\text{category} \arrow[d, no head] & \text{week} \arrow[dr, no head] & & \text{month} \arrow[dl, no head] & \text{state} \arrow[d, no head] \\
\text{pname} & & \text{date} & & \text{city}
\end{tikzcd}$$

Database Management Systems, 2nd Edition. R. Ramakrishnan and J. Gehrke 10



OLAP Queries

- Influenced by SQL and by spreadsheets.
- A common operation is to aggregate a measure over one or more dimensions.
  - Find total sales.
  - Find total sales for each city, or for each state.
  - Find top five products ranked by total sales.
- Roll-up: Aggregating at different levels of a dimension hierarchy.
  - E.g., Given total sales by city, we can roll-up to get sales by state.

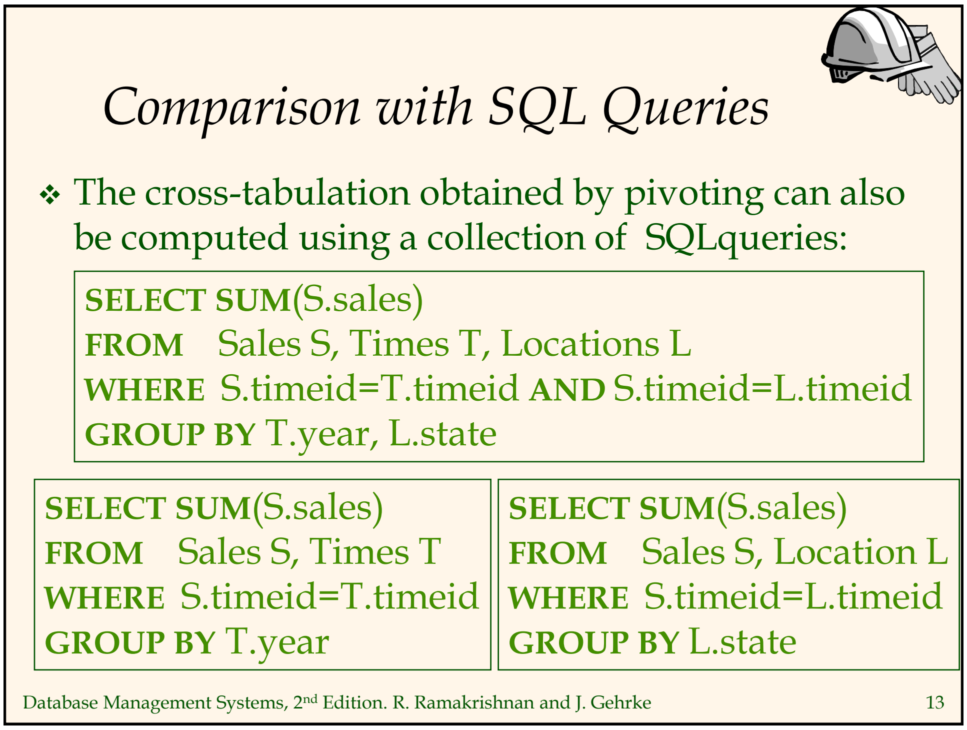


OLAP Queries

- Drill-down: The inverse of roll-up.
  - E.g., Given total sales by state, can drill-down to get total sales by city.
  - E.g., Can also drill-down on different dimension to get total sales by product for each state.
- Pivoting: Aggregation on selected dimensions.
  - E.g., Pivoting on Location and Time yields this **cross-tabulation**:

	WI	CA	Total
1995	63	81	144
1996	38	107	145
1997	75	35	110
Total	176	223	339

- Slicing and Dicing: Equality and range selections on one or more dimensions.



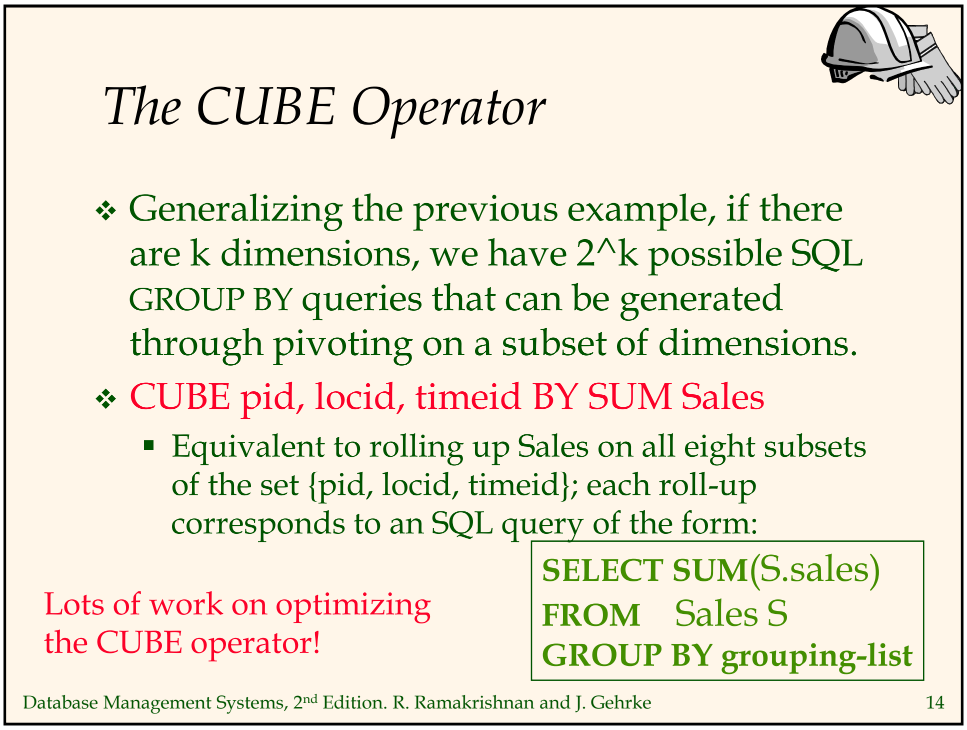
Comparison with SQL Queries

- The cross-tabulation obtained by pivoting can also be computed using a collection of SQLqueries:

```
SELECT SUM(S.sales)
FROM   Sales S, Times T, Locations L
WHERE  S.timeid=T.timeid AND S.timeid=L.timeid
GROUP BY T.year, L.state
```

```
SELECT SUM(S.sales)
FROM   Sales S, Times T
WHERE  S.timeid=T.timeid
GROUP BY T.year
```

```
SELECT SUM(S.sales)
FROM   Sales S, Location L
WHERE  S.timeid=L.timeid
GROUP BY L.state
```



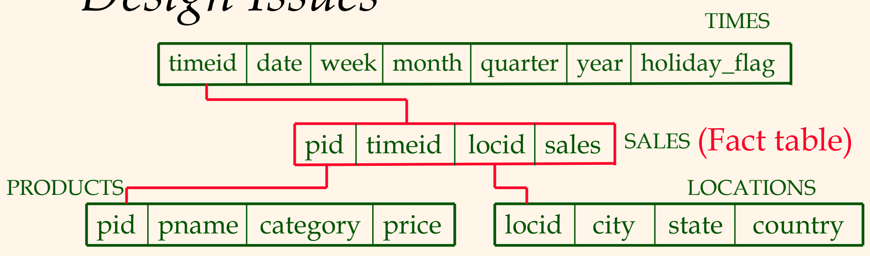
The CUBE Operator

- Generalizing the previous example, if there are k dimensions, we have 2^k possible SQL GROUP BY queries that can be generated through pivoting on a subset of dimensions.
- CUBE pid, locid, timeid BY SUM Sales
  - Equivalent to rolling up Sales on all eight subsets of the set {pid, locid, timeid}; each roll-up corresponds to an SQL query of the form:

```
SELECT SUM(S.sales)
FROM   Sales S
GROUP BY grouping-list
```

Lots of work on optimizing the CUBE operator!

Design Issues


TIMES
timeid | date | week | month | quarter | year | holiday_flag
pid | timeid | locid | sales
SALES (Fact table)
PRODUCTS
pid | pname | category | price
LOCATIONS
locid | city | state | country

- Fact table in BCNF; dimension tables un-normalized.
  - Dimension tables are small; updates/inserts/deletes are rare. So, anomalies less important than query performance.
- This kind of schema is very common in OLAP applications, and is called a star schema; computing the join of all these relations is called a star join.

Implementation Issues

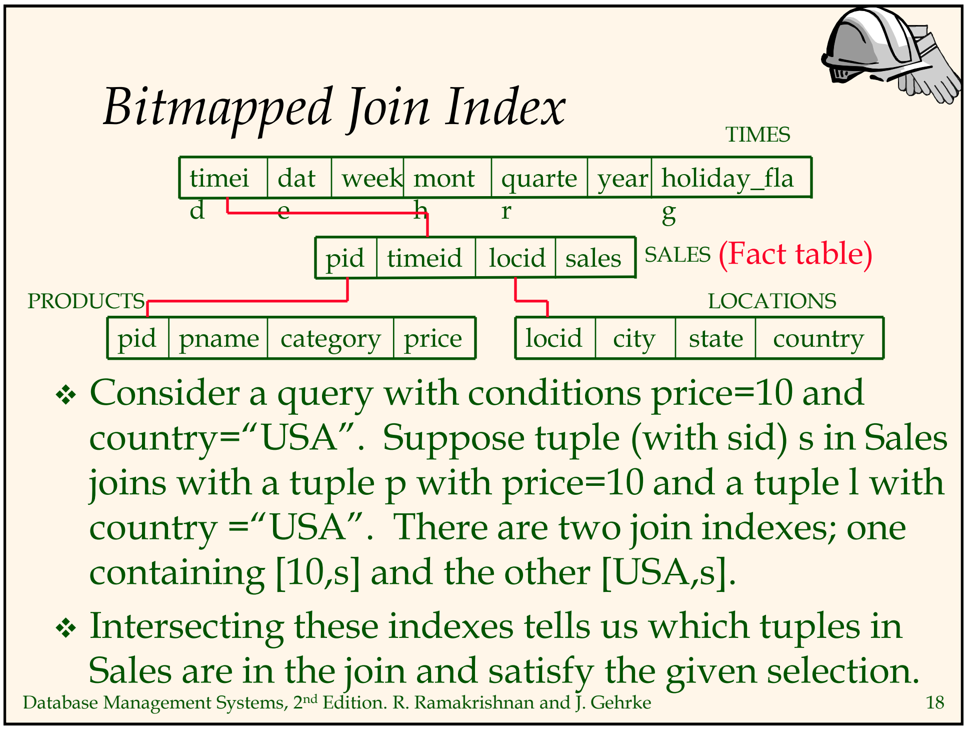
- New indexing techniques: Bitmap indexes, Join indexes, array representations, compression, precomputation of aggregations, etc.
- E.g., Bitmap index:

Bit-vector: **1 bit for each possible value.** F M
Many queries can be answered using bit-vector ops!

sex
10
10
01
10

custid	name	sex	rating
112	Joe	M	3
115	Ram	M	5
119	Sue	F	5
112	Woo	M	4

rating
00100
00001
00001
00010



Bitmapped Join Index

TIMES

timeid	date	week	month	quarter	year	holiday_flag

SALES (Fact table)

pid	timeid	locid	sales

PRODUCTS

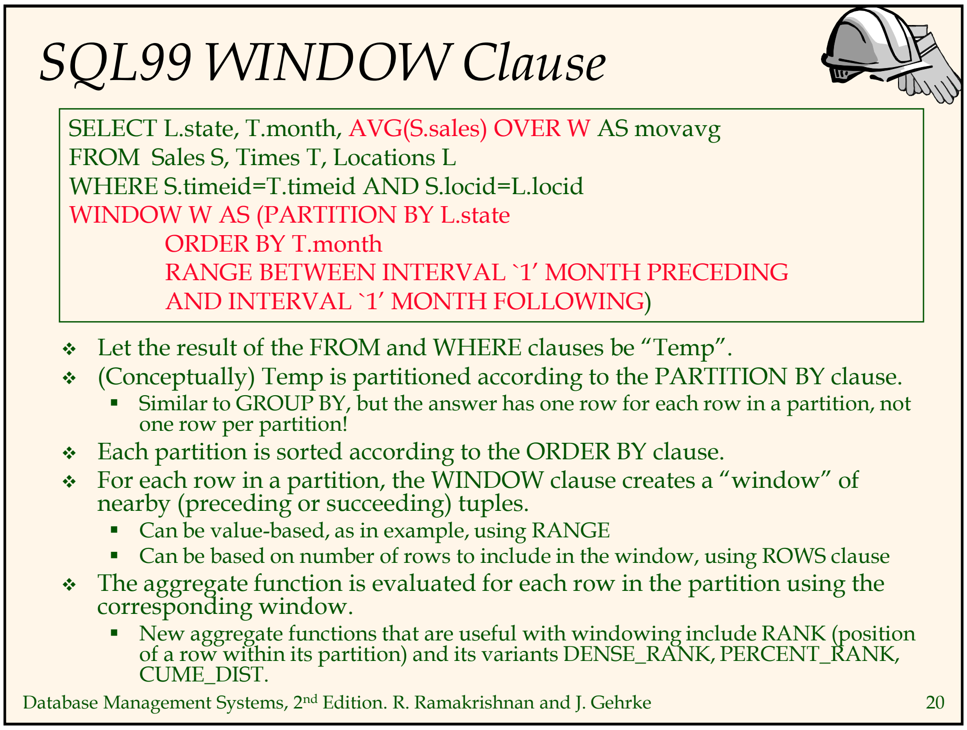
pid	pname	category	price

LOCATIONS

locid	city	state	country

- Consider a query with conditions price=10 and country="USA". Suppose tuple (with sid) s in Sales joins with a tuple p with price=10 and a tuple l with country ="USA". There are two join indexes; one containing [10,s] and the other [USA,s].
- Intersecting these indexes tells us which tuples in Sales are in the join and satisfy the given selection.

Database Management Systems, 2nd Edition. R. Ramakrishnan and J. Gehrke 18

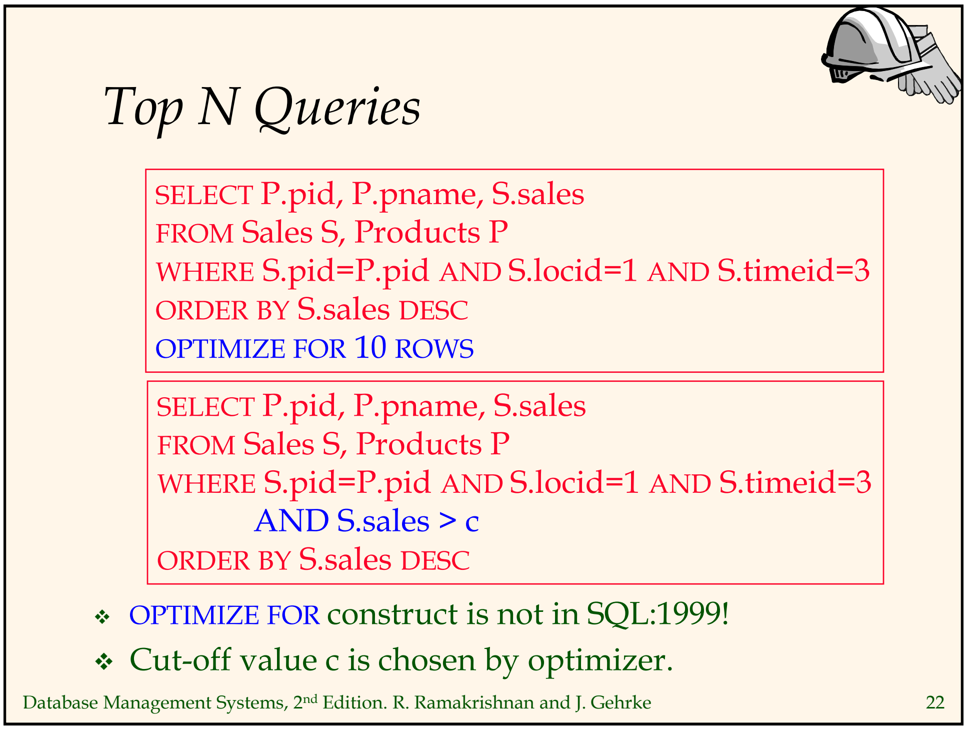


SQL99 WINDOW Clause

```
SELECT L.state, T.month, AVG(S.sales) OVER W AS movavg
FROM  Sales S, Times T, Locations L
WHERE S.timeid=T.timeid AND S.locid=L.locid
WINDOW W AS (PARTITION BY L.state
          ORDER BY T.month
          RANGE BETWEEN INTERVAL `1' MONTH PRECEDING
          AND INTERVAL `1' MONTH FOLLOWING)
```

- Let the result of the FROM and WHERE clauses be "Temp".
- (Conceptually) Temp is partitioned according to the PARTITION BY clause.
  - Similar to GROUP BY, but the answer has one row for each row in a partition, not one row per partition!
- Each partition is sorted according to the ORDER BY clause.
- For each row in a partition, the WINDOW clause creates a "window" of nearby (preceding or succeeding) tuples.
  - Can be value-based, as in example, using RANGE
  - Can be based on number of rows to include in the window, using ROWS clause
- The aggregate function is evaluated for each row in the partition using the corresponding window.
  - New aggregate functions that are useful with windowing include RANK (position of a row within its partition) and its variants DENSE_RANK, PERCENT_RANK, CUME_DIST.

Database Management Systems, 2nd Edition. R. Ramakrishnan and J. Gehrke 20

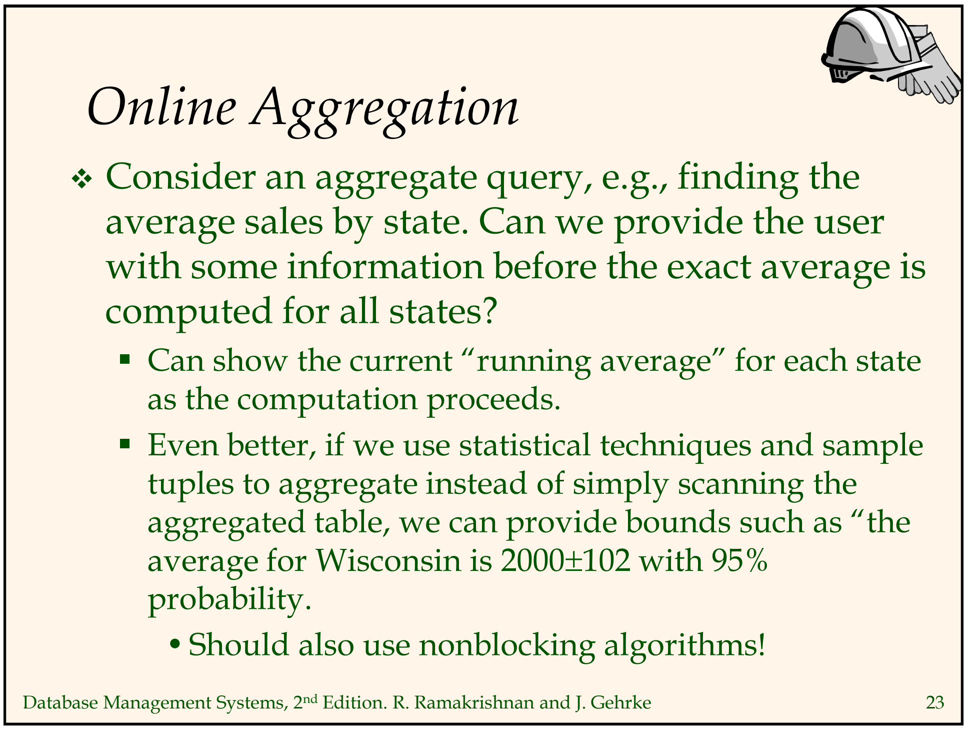


Top N Queries

```
SELECT P.pid, P.pname, S.sales
FROM Sales S, Products P
WHERE S.pid=P.pid AND S.locid=1 AND S.timeid=3
ORDER BY S.sales DESC
OPTIMIZE FOR 10 ROWS
```

```
SELECT P.pid, P.pname, S.sales
FROM Sales S, Products P
WHERE S.pid=P.pid AND S.locid=1 AND S.timeid=3
          AND S.sales > c
ORDER BY S.sales DESC
```

- OPTIMIZE FOR construct is not in SQL:1999!
- Cut-off value c is chosen by optimizer.



Online Aggregation

- Consider an aggregate query, e.g., finding the average sales by state. Can we provide the user with some information before the exact average is computed for all states?
  - Can show the current "running average" for each state as the computation proceeds.
  - Even better, if we use statistical techniques and sample tuples to aggregate instead of simply scanning the aggregated table, we can provide bounds such as "the average for Wisconsin is 2000±102 with 95% probability.
    - Should also use nonblocking algorithms!

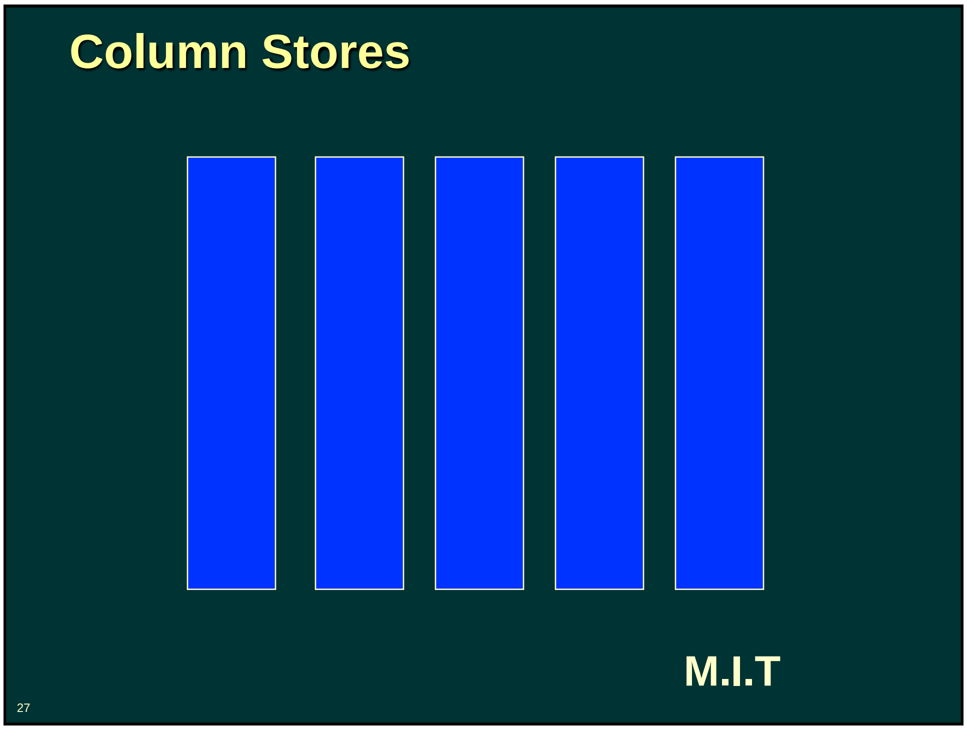
Data Warehousing: Practice

- "Options" to commercial RDBMS
  - Turn off logging, locking
  - Large blocks, no empty space
  - Bitmap indices, data cubes
- Specialized database
  - No "dead code" for Transaction features
  - Compressed representations (more in memory)
  - …

C-Store: A Column-oriented DBMS

By

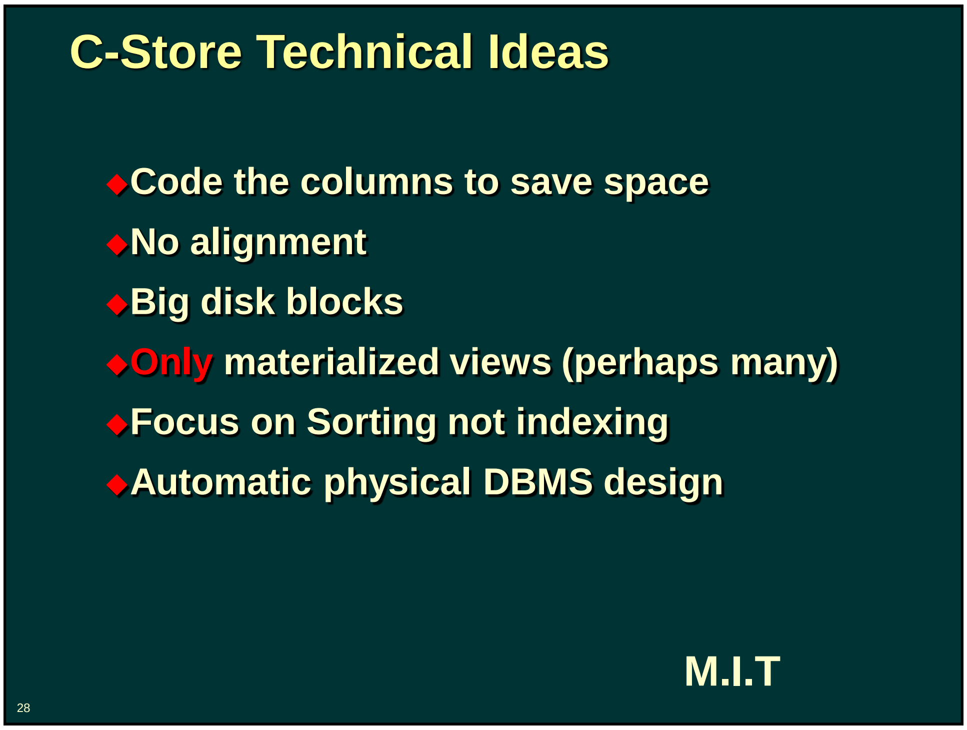
New England Database Group



Column Stores

M.I.T

27

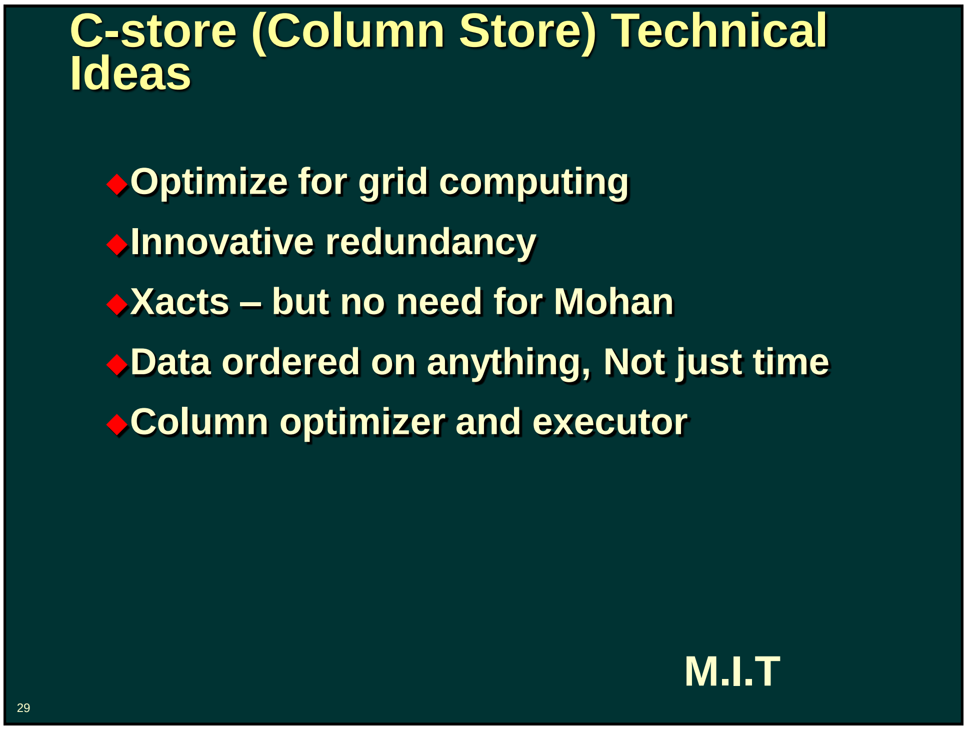


C-Store Technical Ideas

- Code the columns to save space
- No alignment
- Big disk blocks
- Only materialized views (perhaps many)
- Focus on Sorting not indexing
- Automatic physical DBMS design

M.I.T

28

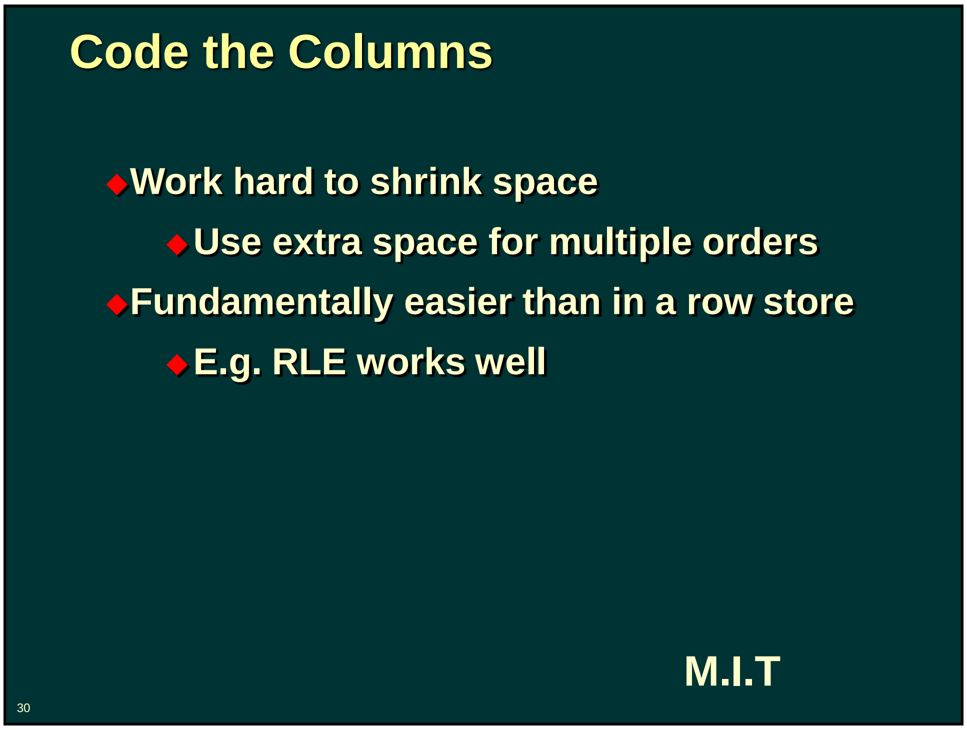


C-store (Column Store) Technical Ideas

- Optimize for grid computing
- Innovative redundancy
- Xacts – but no need for Mohan
- Data ordered on anything, Not just time
- Column optimizer and executor

M.I.T

29

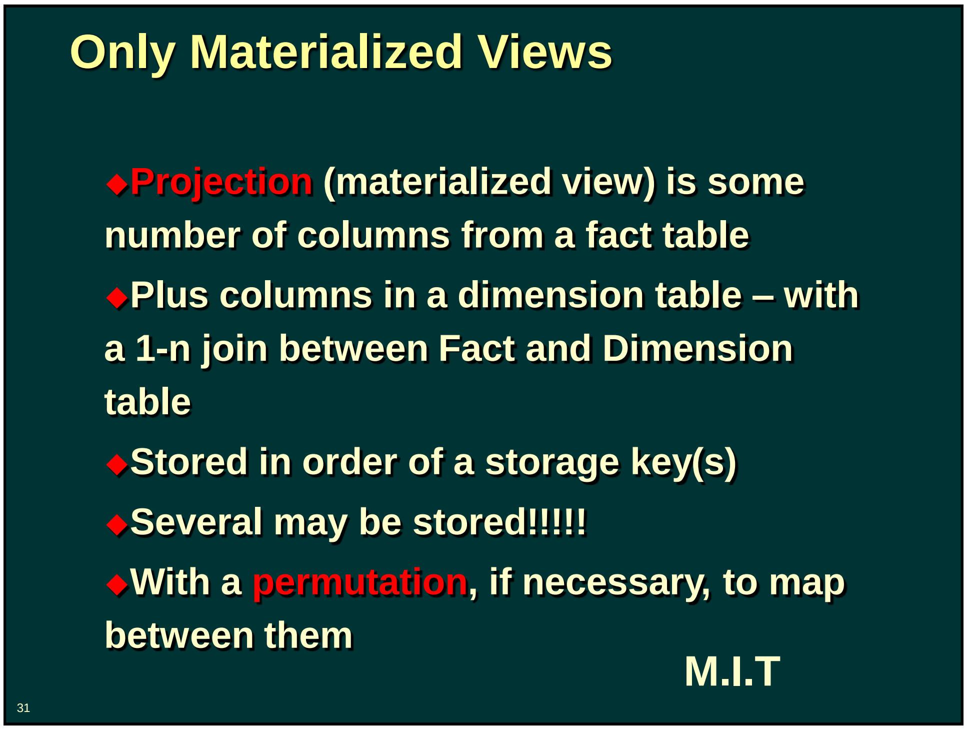


Code the Columns

- Work hard to shrink space
  - Use extra space for multiple orders
- Fundamentally easier than in a row store
  - E.g. RLE works well

M.I.T

30

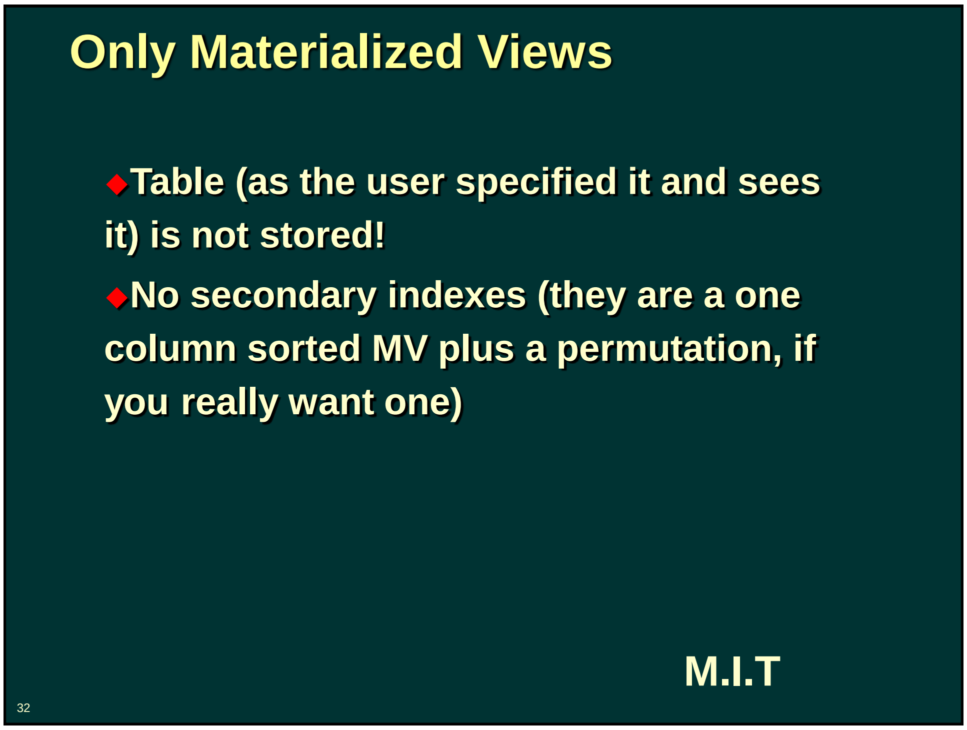


Only Materialized Views

- Projection (materialized view) is some number of columns from a fact table
- Plus columns in a dimension table – with a 1-n join between Fact and Dimension table
- Stored in order of a storage key(s)
- Several may be stored!!!!!
- With a permutation, if necessary, to map between them

M.I.T

31

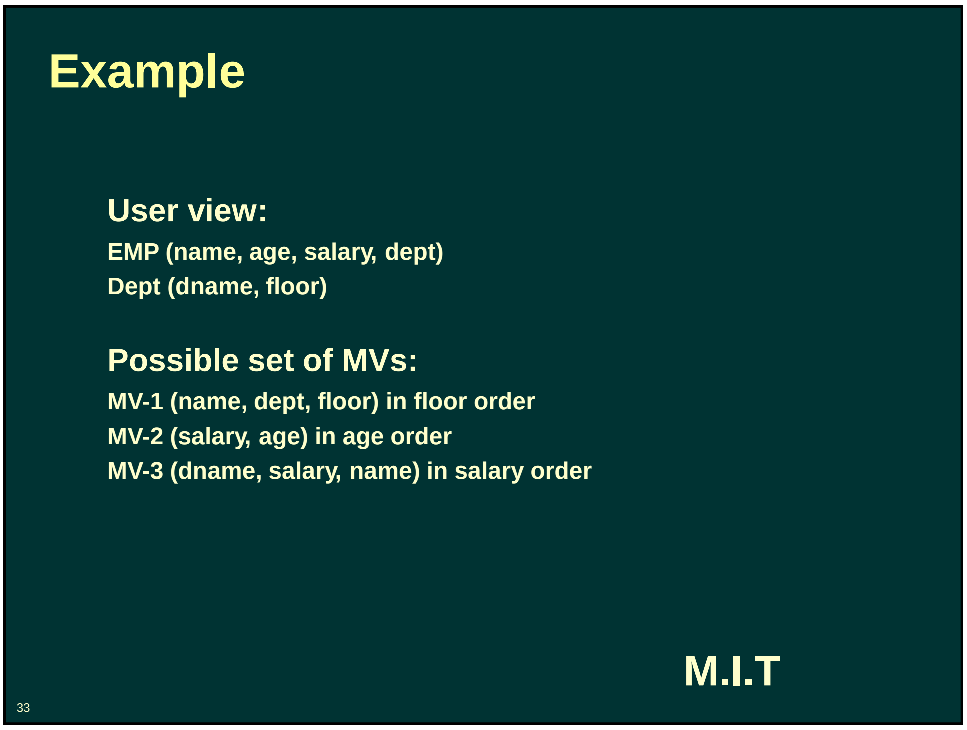


Only Materialized Views

- Table (as the user specified it and sees it) is not stored!
- No secondary indexes (they are a one column sorted MV plus a permutation, if you really want one)

M.I.T

32



Example

User view:

EMP (name, age, salary, dept)

Dept (dname, floor)

Possible set of MVs:

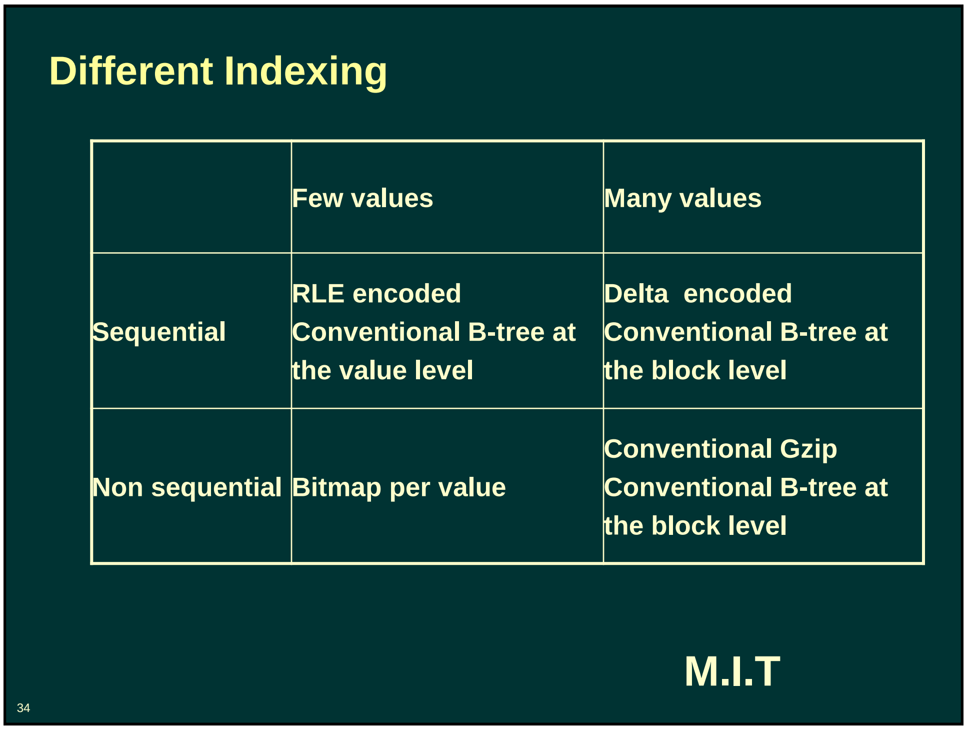
MV-1 (name, dept, floor) in floor order

MV-2 (salary, age) in age order

MV-3 (dname, salary, name) in salary order

M.I.T

33

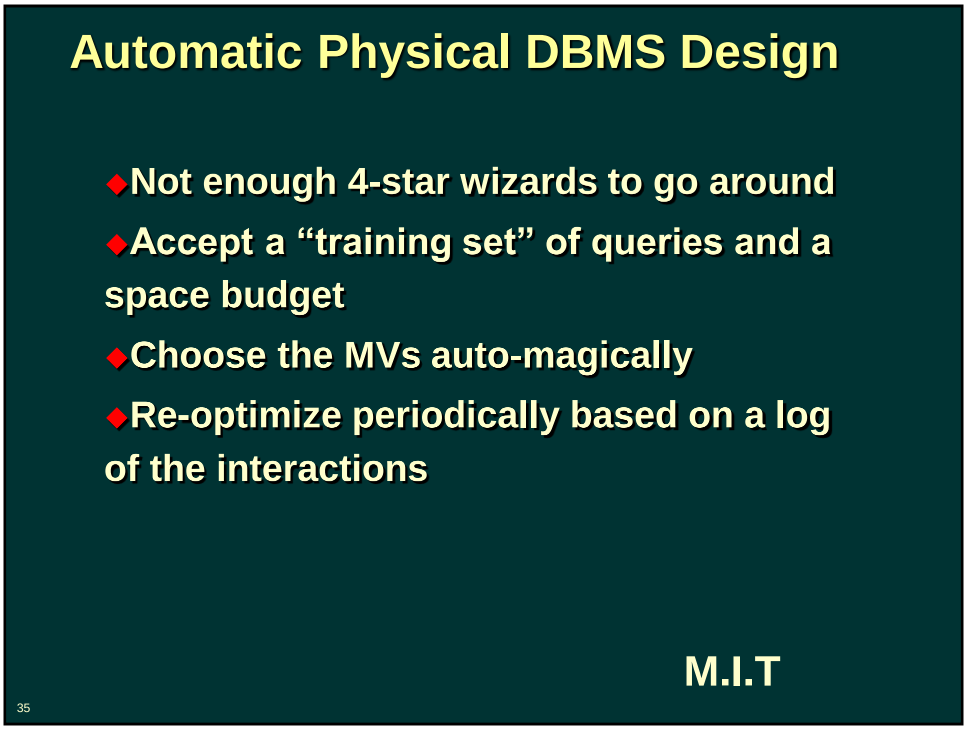


Different Indexing

	Few values	Many values
Sequential	RLE encoded Conventional B-tree at the value level	Delta encoded Conventional B-tree at the block level
Non sequential	Bitmap per value	Conventional Gzip Conventional B-tree at the block level

M.I.T

34

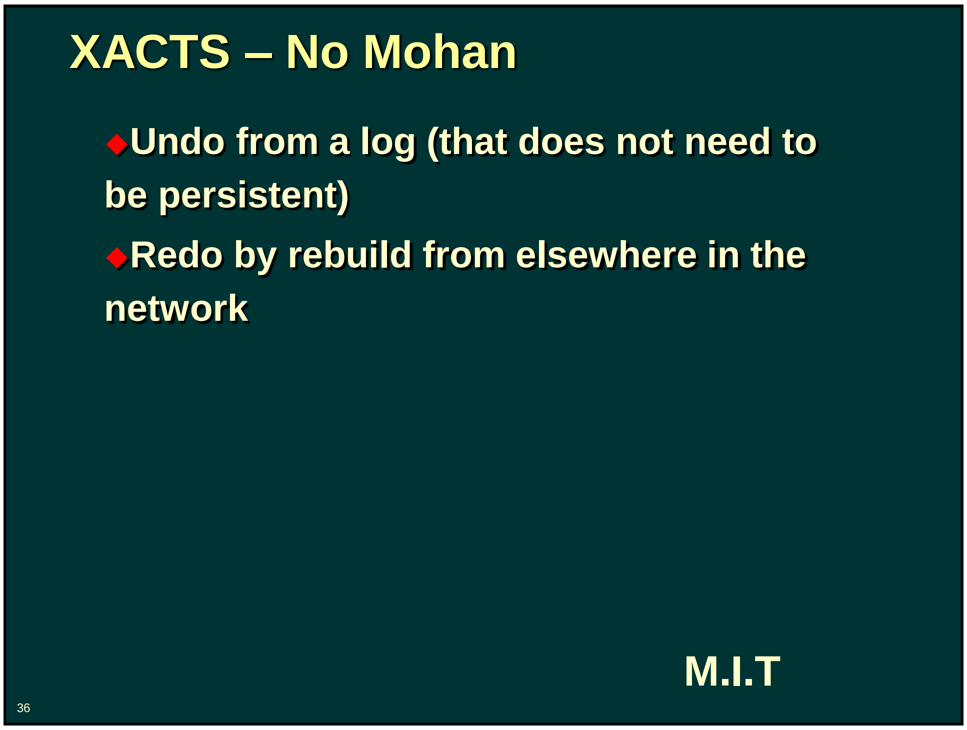


Automatic Physical DBMS Design

- Not enough 4-star wizards to go around
- Accept a “training set” of queries and a space budget
- Choose the MVs auto-magically
- Re-optimize periodically based on a log of the interactions

M.I.T

35

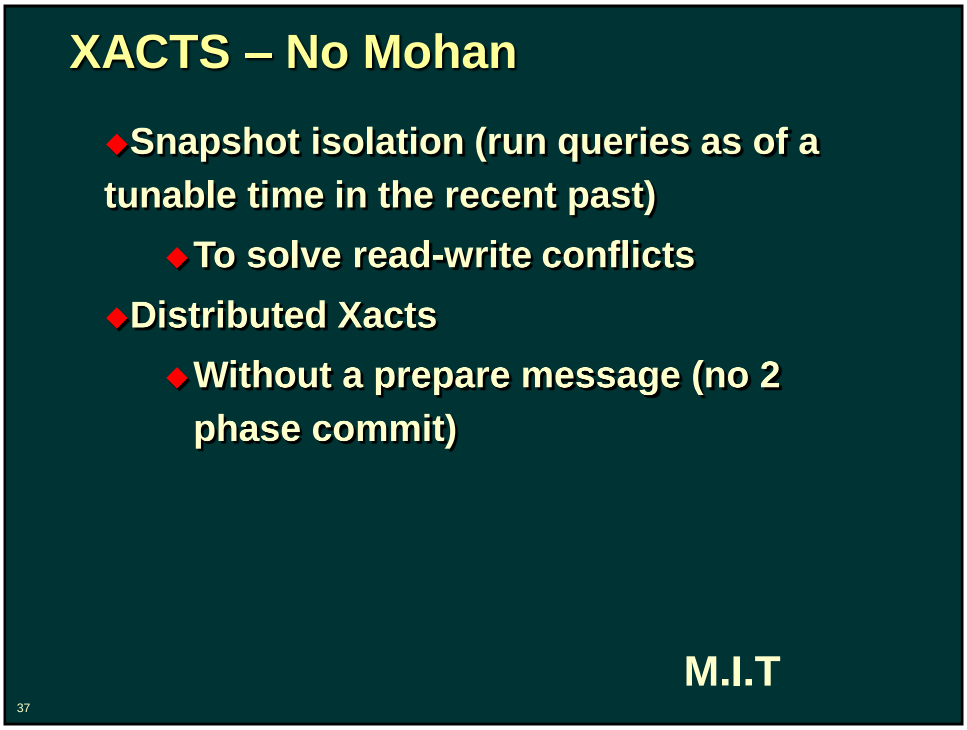


XACTS – No Mohan

- Undo from a log (that does not need to be persistent)
- Redo by rebuild from elsewhere in the network

M.I.T

36

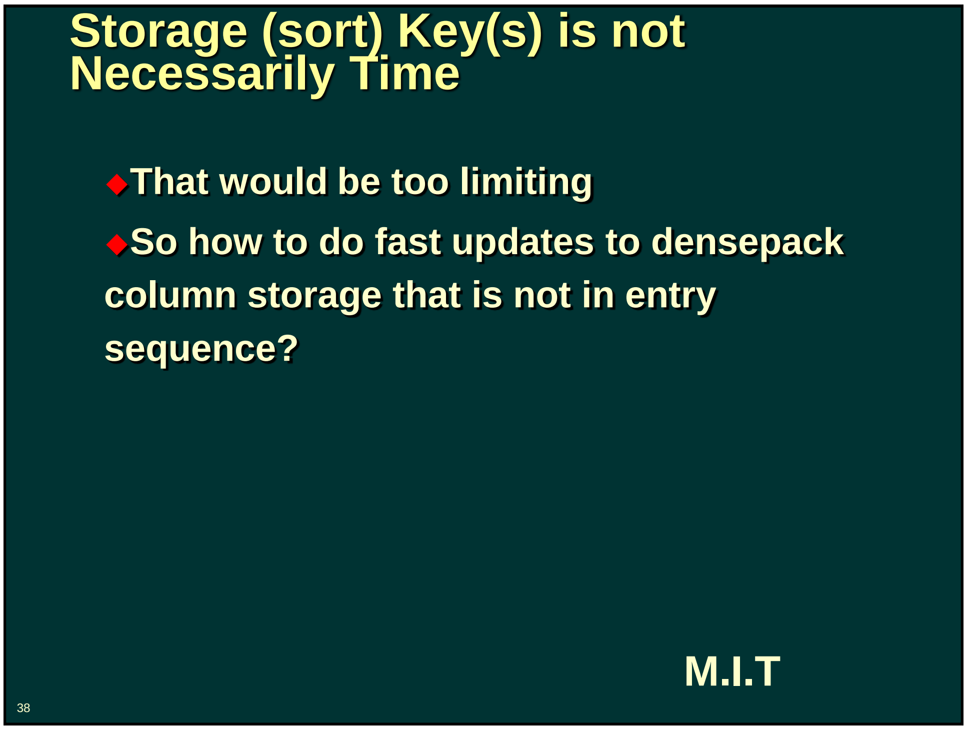


XACTS – No Mohan

- Snapshot isolation (run queries as of a tunable time in the recent past)
  - To solve read-write conflicts
- Distributed Xacts
  - Without a prepare message (no 2 phase commit)

M.I.T

37



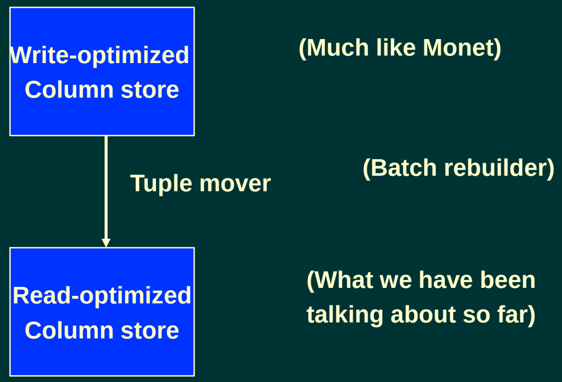
Storage (sort) Key(s) is not Necessarily Time

- That would be too limiting
- So how to do fast updates to densepack column storage that is not in entry sequence?

M.I.T

38

Solution – a Hybrid Store


Write-optimized Column store
(Much like Monet)
Tuple mover
(Batch rebuilder)
Read-optimized Column store
(What we have been talking about so far)

M.I.T

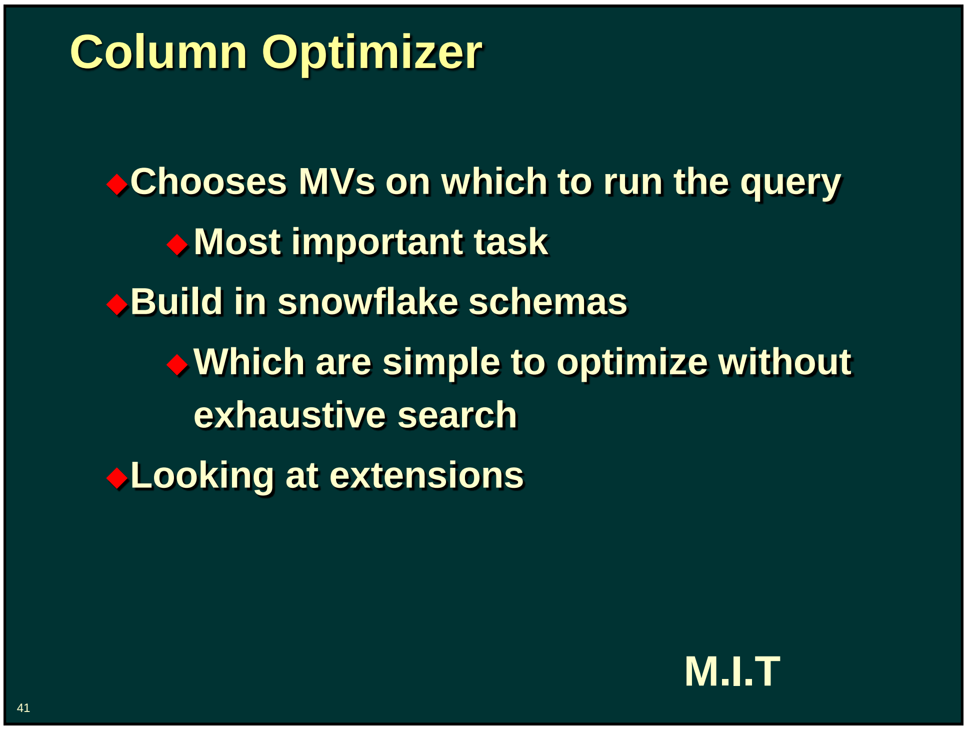
39

Column Executor

- Column operations – not row operations
- Columns remain coded – if possible
- Late materialization of columns

M.I.T

40

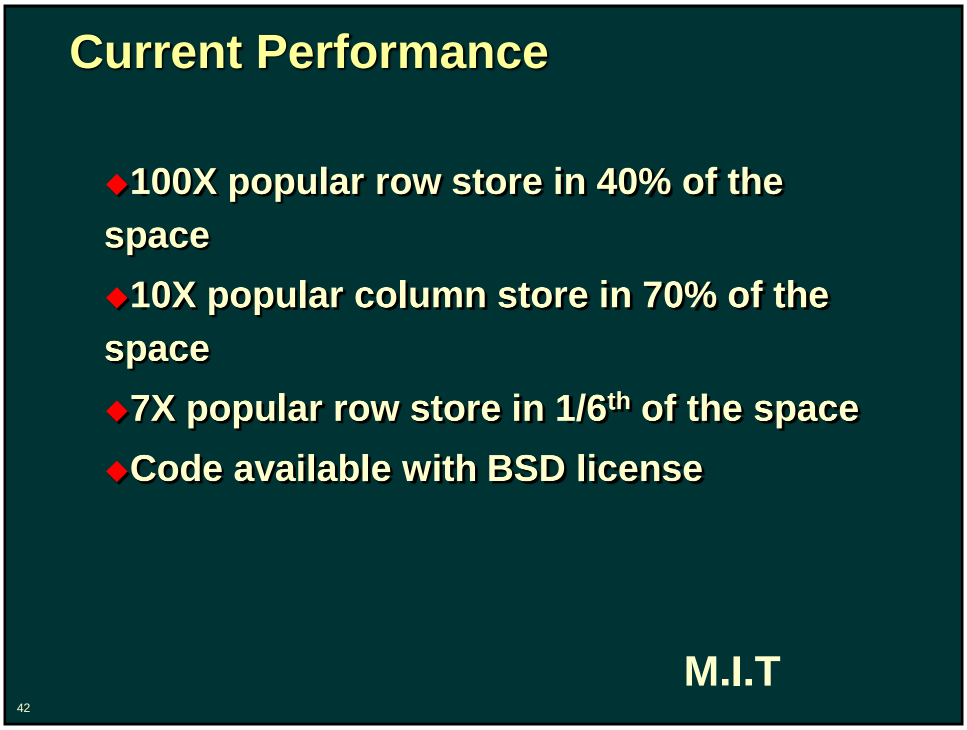


Column Optimizer

- Chooses MVs on which to run the query
  - Most important task
- Build in snowflake schemas
  - Which are simple to optimize without exhaustive search
- Looking at extensions

M.I.T

41

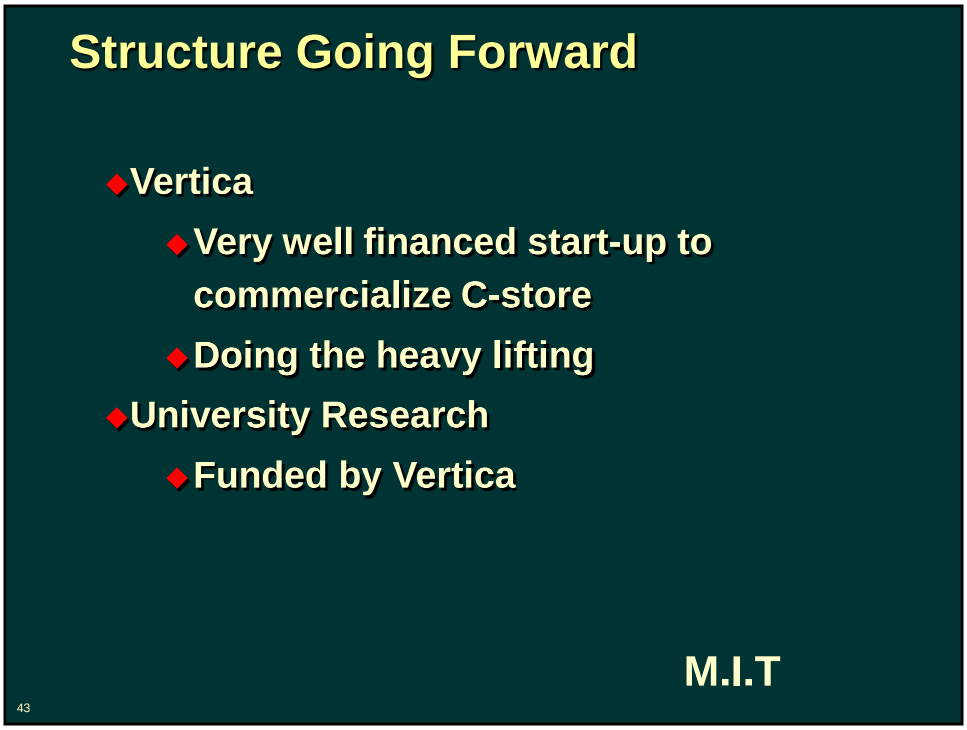


Current Performance

- 100X popular row store in 40% of the space
- 10X popular column store in 70% of the space
- 7X popular row store in 1/6th of the space
- Code available with BSD license

M.I.T

42



Structure Going Forward

- Vertica
  - Very well financed start-up to commercialize C-store
  - Doing the heavy lifting
- University Research
  - Funded by Vertica

M.I.T

43