Microsoft tech·ed North America 2011

MAY 16-19, 2011 | ATLANTA

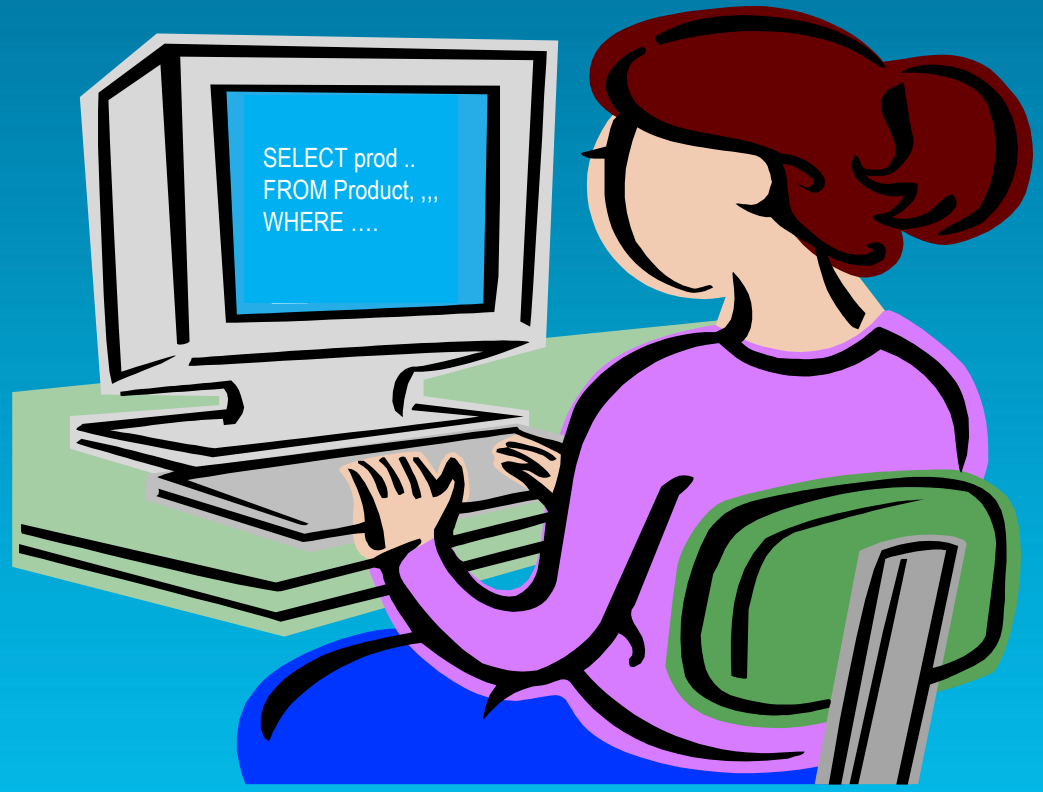
新一代面向数据仓库索引技术概览

孙巍
技术总监
北京中达金桥技术服务有限公司

Alice, Today

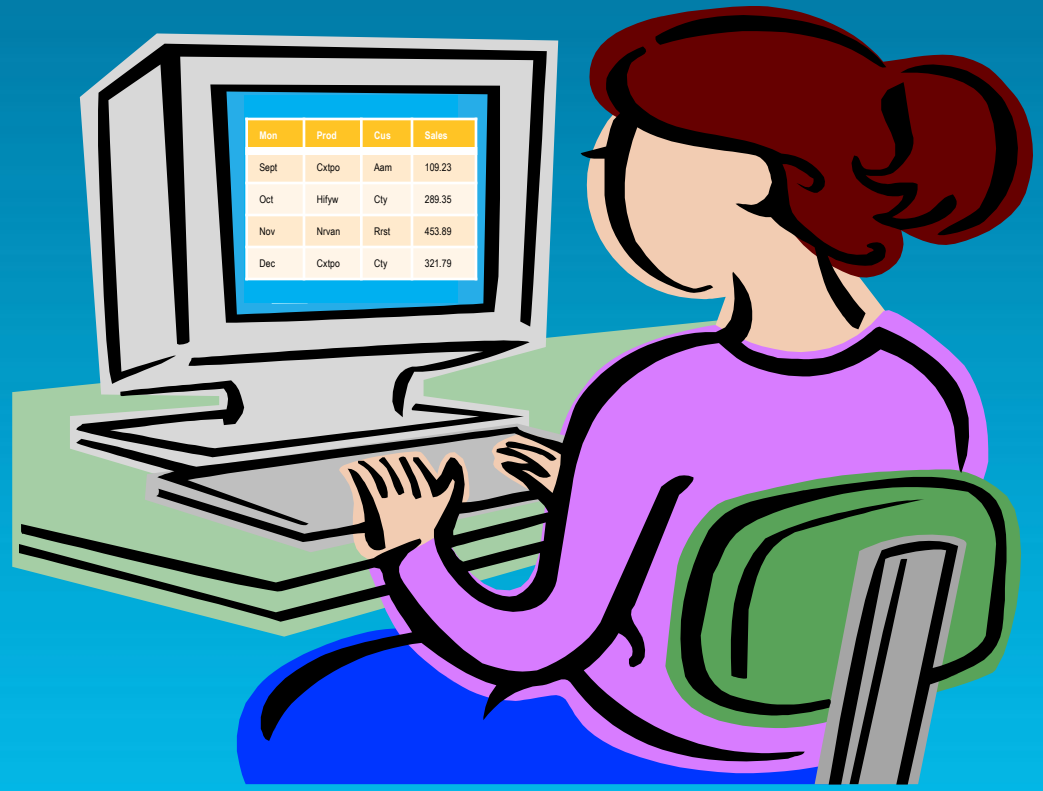
Alice, Today

Alice, Today



Alice, Today

Alice, Today

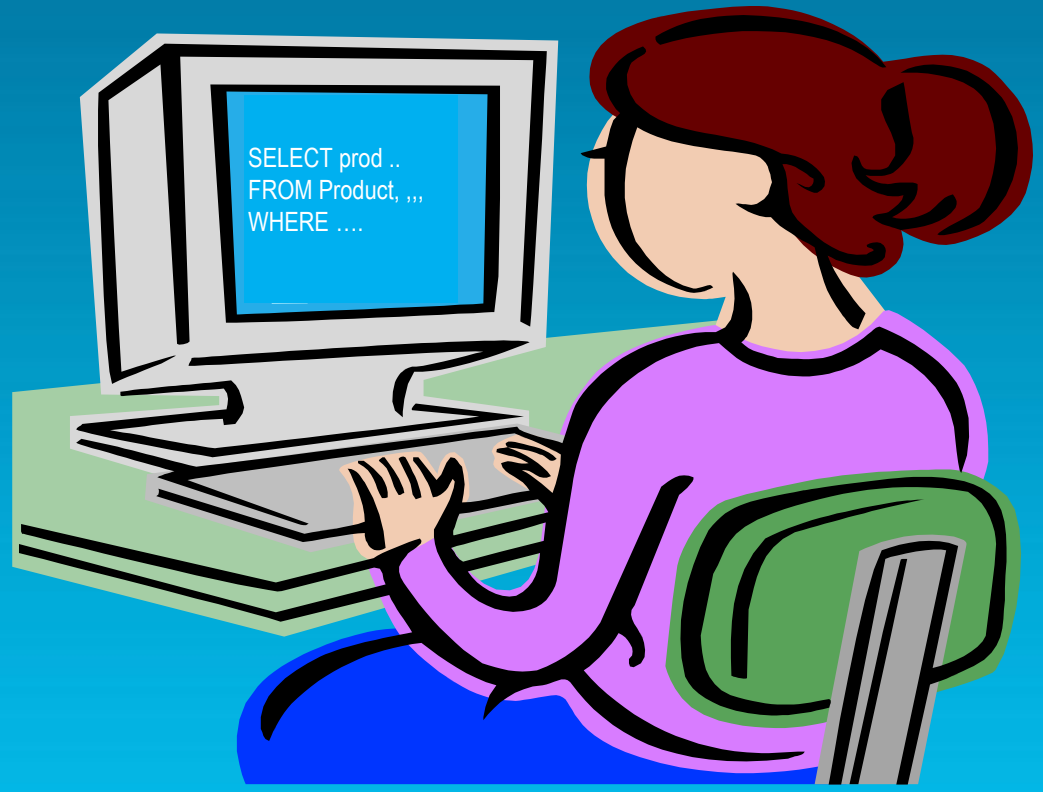

Mon
Prod
Cus
Sales
Sept
Cxtpo
Aam
109.23
Oct
Hifyw
Cty
289.35
Nov
Nrvan
Rrst
453.89
Dec
Cxtpo
Cty
321.79

Alice, Today

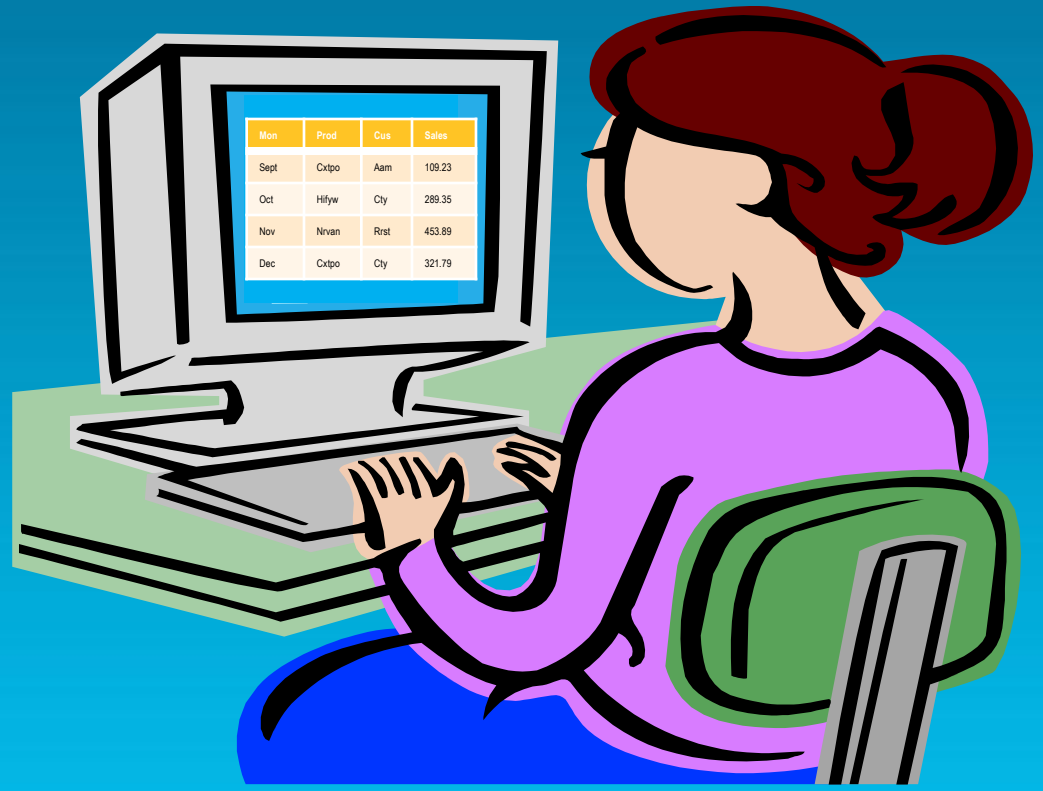

Mon
Prod
Cus
Sales
Sept
Cxtpo
Aam
109.23
Oct
Hifyw
Cty
289.35
Nov
Nrvan
Rrst
453.89
Dec
Cxtpo
Cty
321.79

Alice, Tomorrow

Alice, Tomorrow



Alice, Tomorrow



Alice, Tomorrow

Alice, Tomorrow

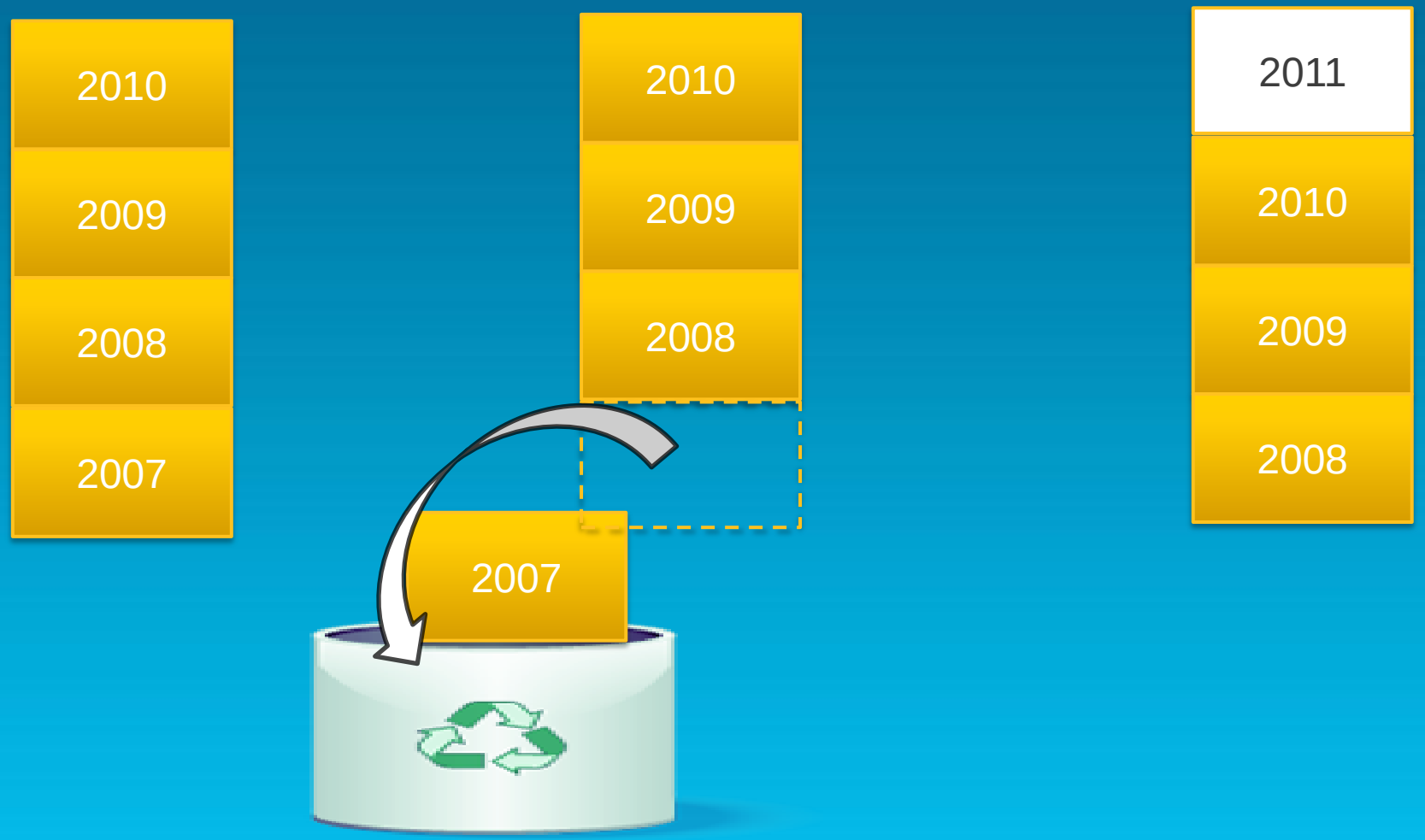
Project Apollo

- Accelerates data warehouse queries
  - New columnstore index
  - Improved query execution
- Makes SQL Server more interactive
- Easy to use
- Reduces TCO

Data warehouse workload

- Read-mostly
  - Load large amounts of data
  - Append new data incrementally
  - Rarely update existing data
  - Often retain data for given time (e.g. 1, 3, 7 yrs)
    - Sliding window data management

Sliding window



Data warehouse schemas and queries

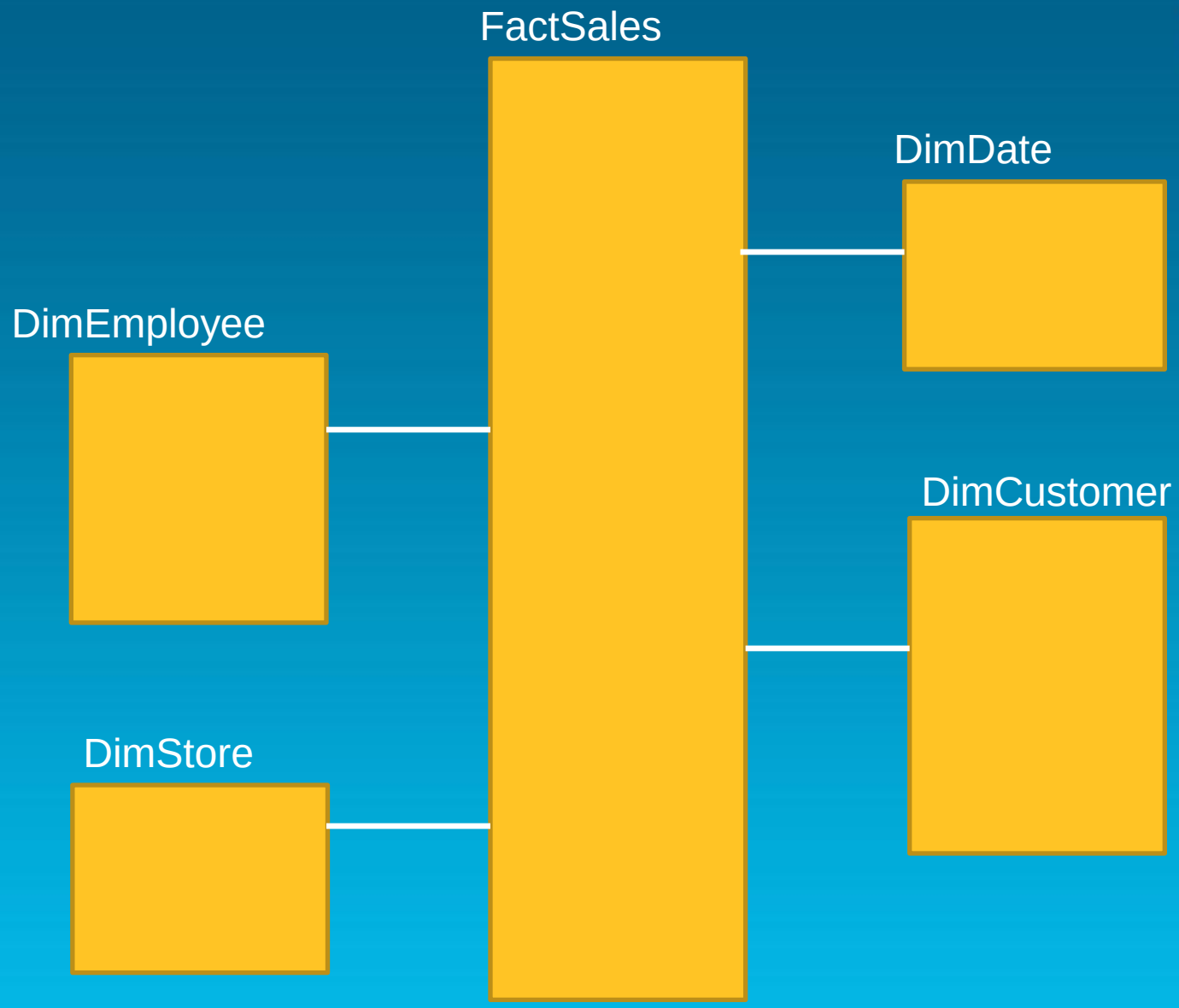
- Star schema
  - Fact tables ← put columnstore indexes here
  - Dimension tables
- Star joins
- Most queries aggregate data

Star schema

FactSales(CustomerKey int,
ProductKey int,
EmployeeKey int,
StoreKey int,
OrderDateKey int,
SalesAmount money)

DimCustomer(CustomerKey int,
FirstName nvarchar(50),
LastName nvarchar(50),
Birthdate date,
EmailAddress nvarchar(50))

DimProduct …



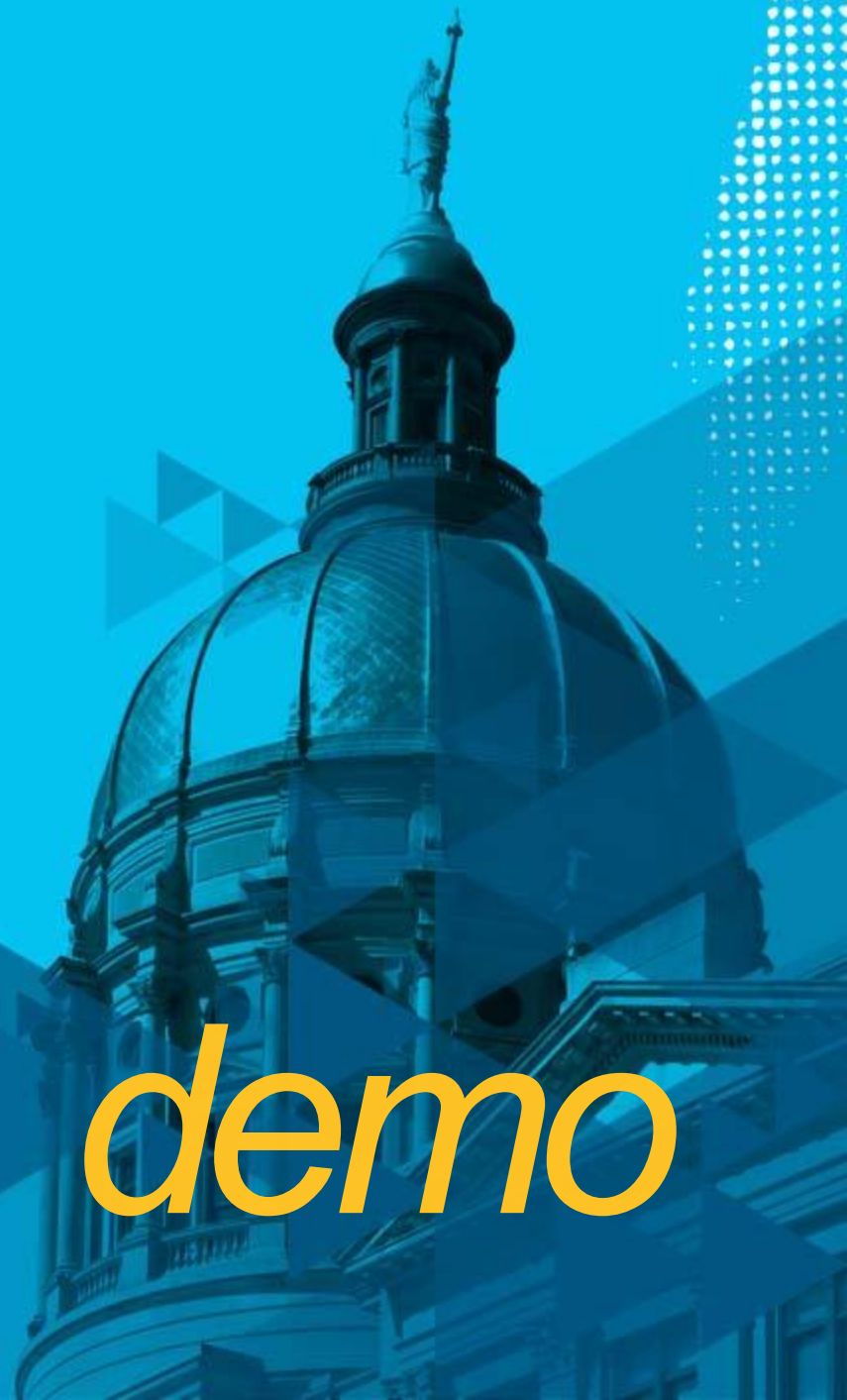
Star join query

```
SELECT TOP 10 p.ModelName, p.EnglishDescription,
      SUM(f.SalesAmount) as SalesAmount
FROM FactResellerSalesPart f, DimProduct p, DimEmployee e
WHERE f.ProductKey=p.ProductKey
      AND e.EmployeeKey=f.EmployeeKey
      AND f.OrderDateKey >= 20030601
      AND p.ProductLine = 'M' -- Mountain
      AND p.ModelName LIKE '%Frame%'
      AND e.SalesTerritoryKey = 1
GROUP BY p.ModelName, p.EnglishDescription
ORDER BY SUM(f.SalesAmount) desc;
```

“Typical” data warehouse queries

- Process large amounts of data
- Reporting queries
- Often slow (minutes to hours)
- DBAs spend considerable effort
  - Designing indexes, tuning queries
  - Building summary tables, indexed views, OLAP cubes

Query Acceleration With Columnstore Indexes

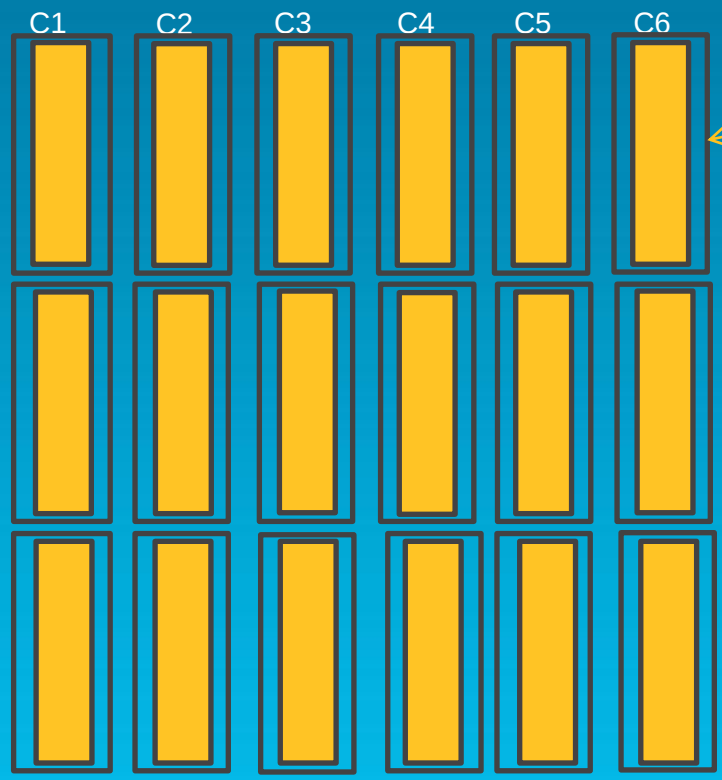


demo

Columnar storage structure

Row store:

Column store:

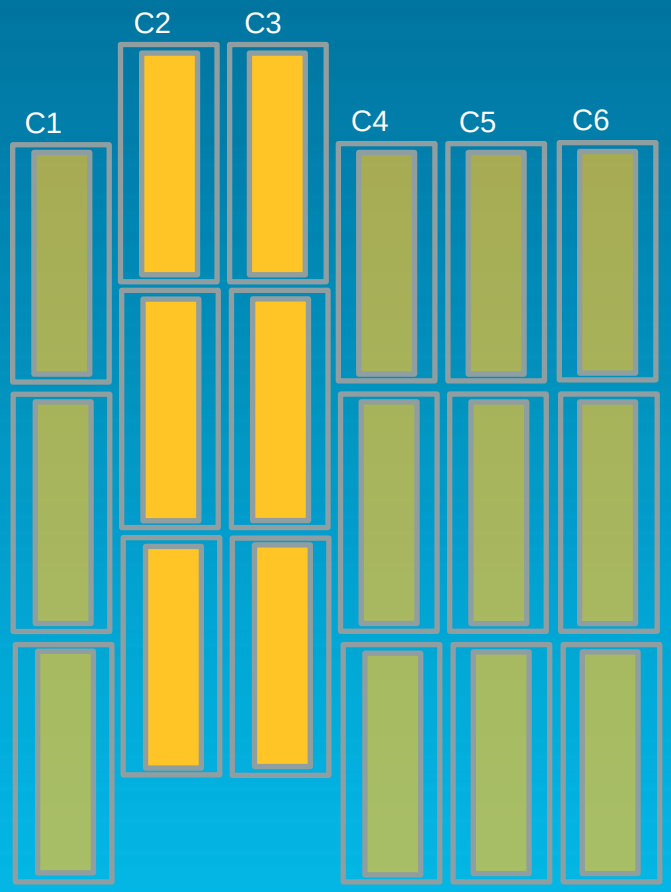


Pages

Uses VertiPaq compression

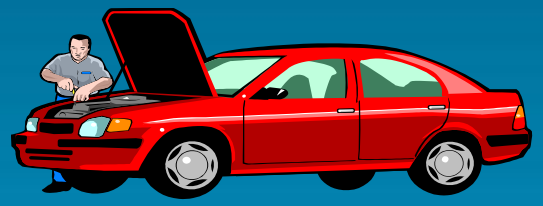
Reduced I/O using columnstore indexes

SELECT region, sum (sales) …



- Fetches only needed columns from disk
- Columns are compressed
- Less IO
- Better buffer hit rates

Advanced query execution technology



- Batch mode execution of some operations
  - Processes rows in batches
  - Groups of batch operations in query plan
- Efficient data representation
- Highly efficient algorithms
- Better parallelism

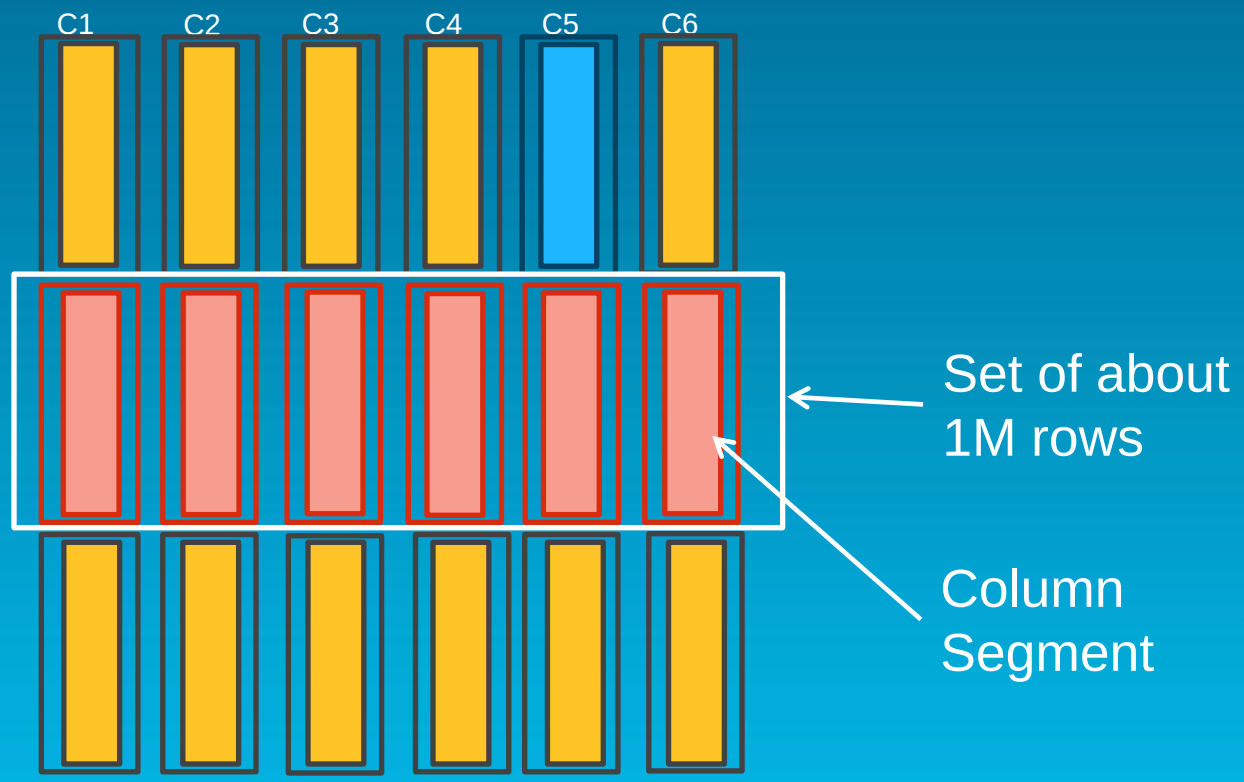
Batch Mode: Performance Gains

- Depends on query, data characteristics, etc.
  - Limited by how much of query plan uses columnstore & batch processing; may achieve 100x speed up
- 1 TB version of TPC-DS DB 32 proc, 256 GB RAM

```
SELECT w_city, w_state, d_year, SUM(cs_sales_price) AS cs_sales_price
FROM warehouse, catalog_sales, date_dim
WHERE w_warehouse_sk = cs_warehouse_sk and cs_sold_date_sk = d_date_sk
     and w_state = 'SD' and d_year = 2002
GROUP BY w_city, w_state, d_year
ORDER BY d_year, w_state, w_city;
```

	Cold Buffer Pool		Warm Buffer Pool	
	CPU	**Elapsed**	**CPU**	**Elapsed**
Row store	259 s	20 s	206 s	3.1 s
Columnstore + batch	19.8 s	0.8 s	16.3 s	0.3 s
Speedup	13 X	25 X	13 X	10 X

Column segments



- Column segment contains values from one column for a set of about 1M rows
- Column segments are compressed
- Each column segment stored in separate LOB
- Column segment is unit of transfer from disk

Constraints on use with other indexes & partitioning

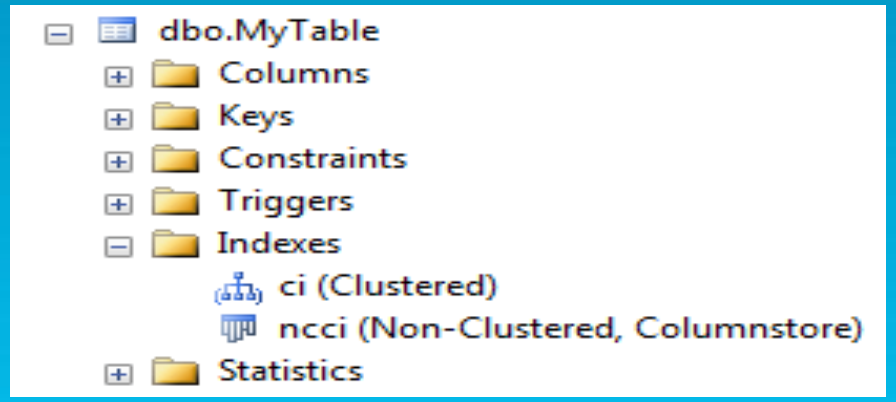
- Base table must be clustered B-tree or heap
- Columnstore index:
  - nonclustered
  - one per table
  - must be partition-aligned
  - not allowed on indexed view
  - can’t be a filtered index

Creating the columnstore index

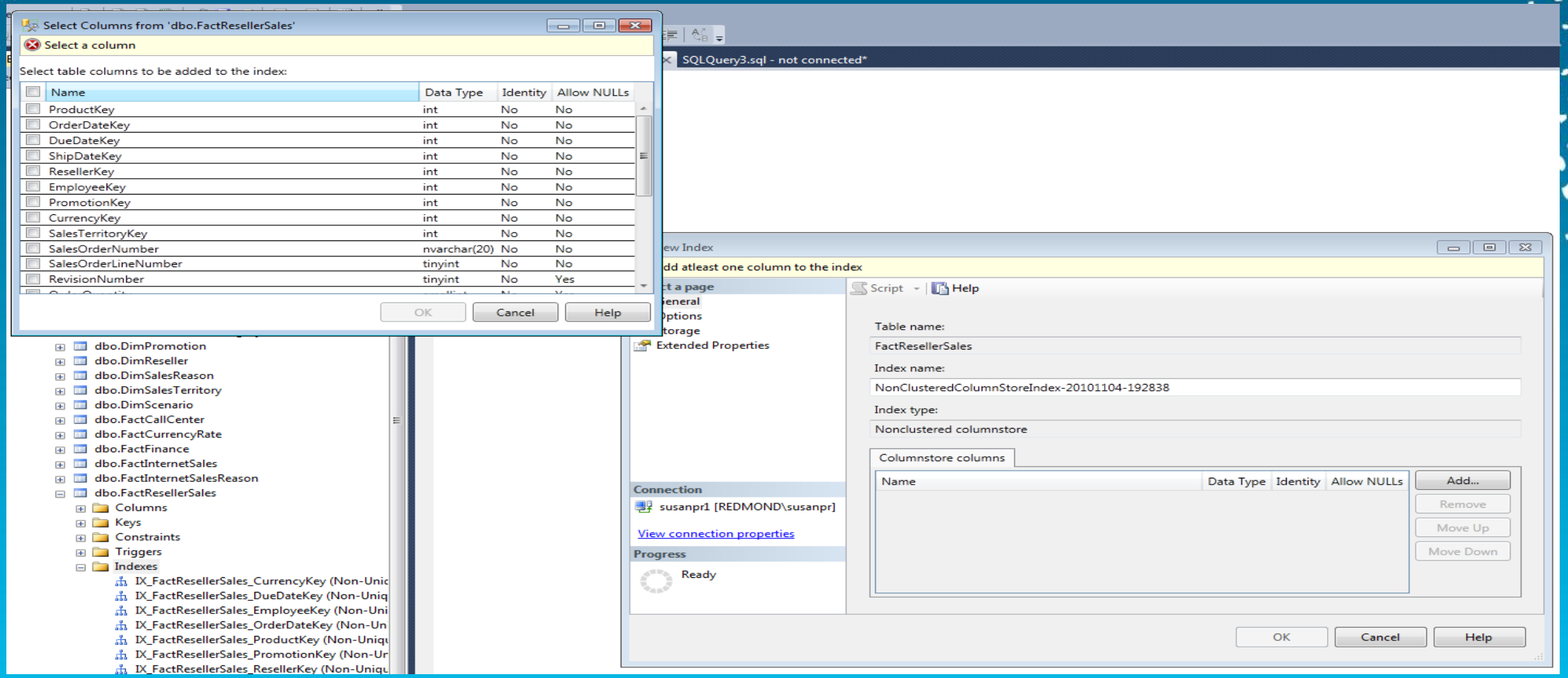
- Create the table
- Load data into the table
- Create a non-clustered columnstore index on all, or some, columns

```
CREATE NONCLUSTERED COLUMNSTORE INDEX ncci ON myTable(OrderDate, ProductID,
SaleAmount)
```

Object Explorer



Index dialog makes index creation easy



Query optimization with columnstores

- Cost-based
- Optimizer chooses index
- Optimizer chooses batch mode operators

Using hints with columnstore indexes

- use the columnstore index

```
select distinct (SalesTerritoryKey)
from dbo.FactResellerSales with (index (ncci))
```

- use a different index

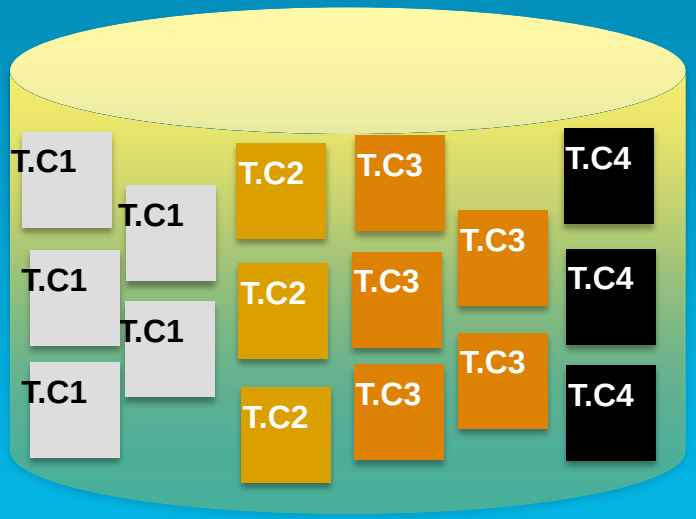
```
select distinct (SalesTerritoryKey)
from dbo.FactResellerSales with (index (ci))
```

- ignore columnstore

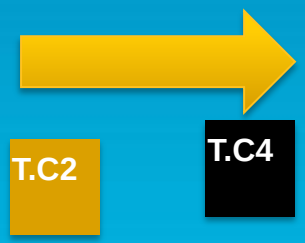
```
select distinct (SalesTerritoryKey)
from dbo.FactResellerSales
option(ignore_nonclustered_columnstore_index)
```

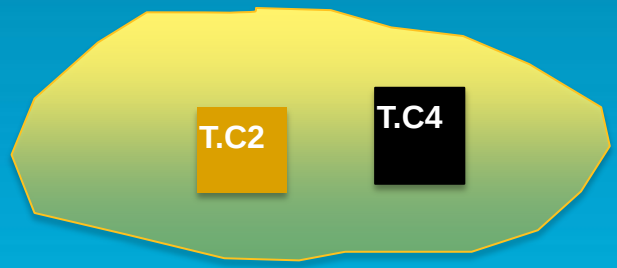
Memory management

- Memory management is automatic
- Columnstore persisted on disk
- Needed columns fetched into memory
- Column segments flow between disk and memory

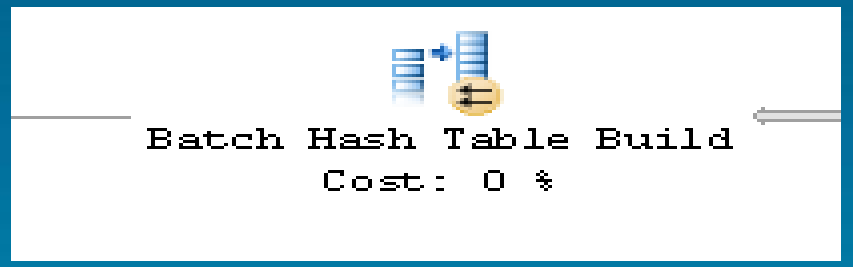


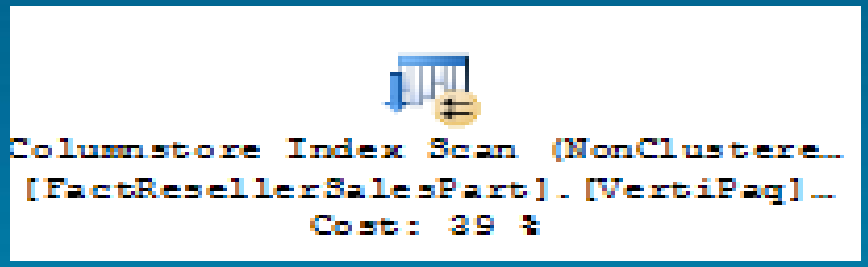

```
SELECT C2, SUM(C4)
FROM T
GROUP BY C2;
```

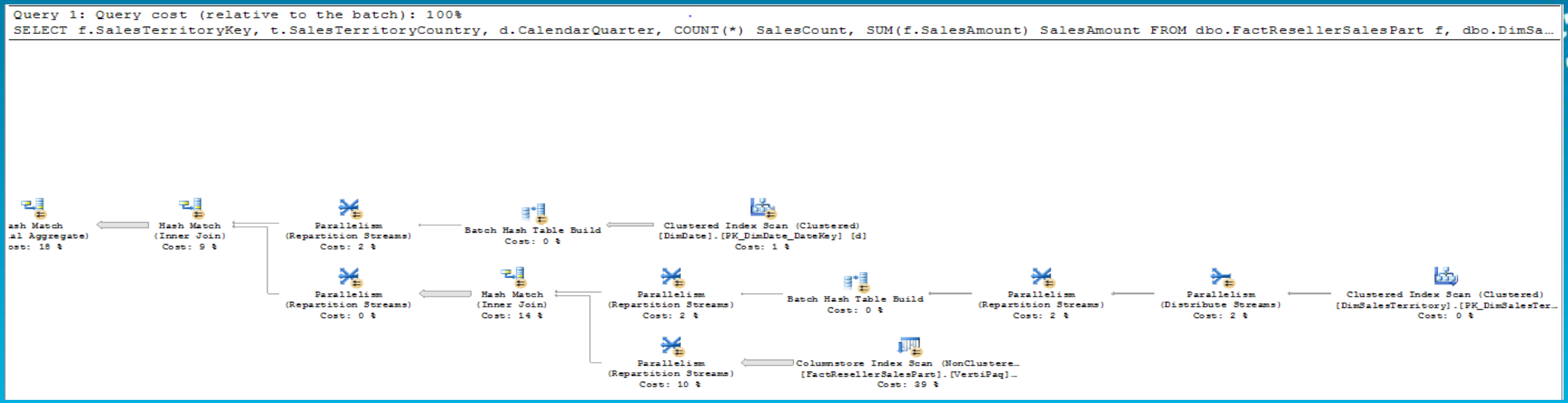


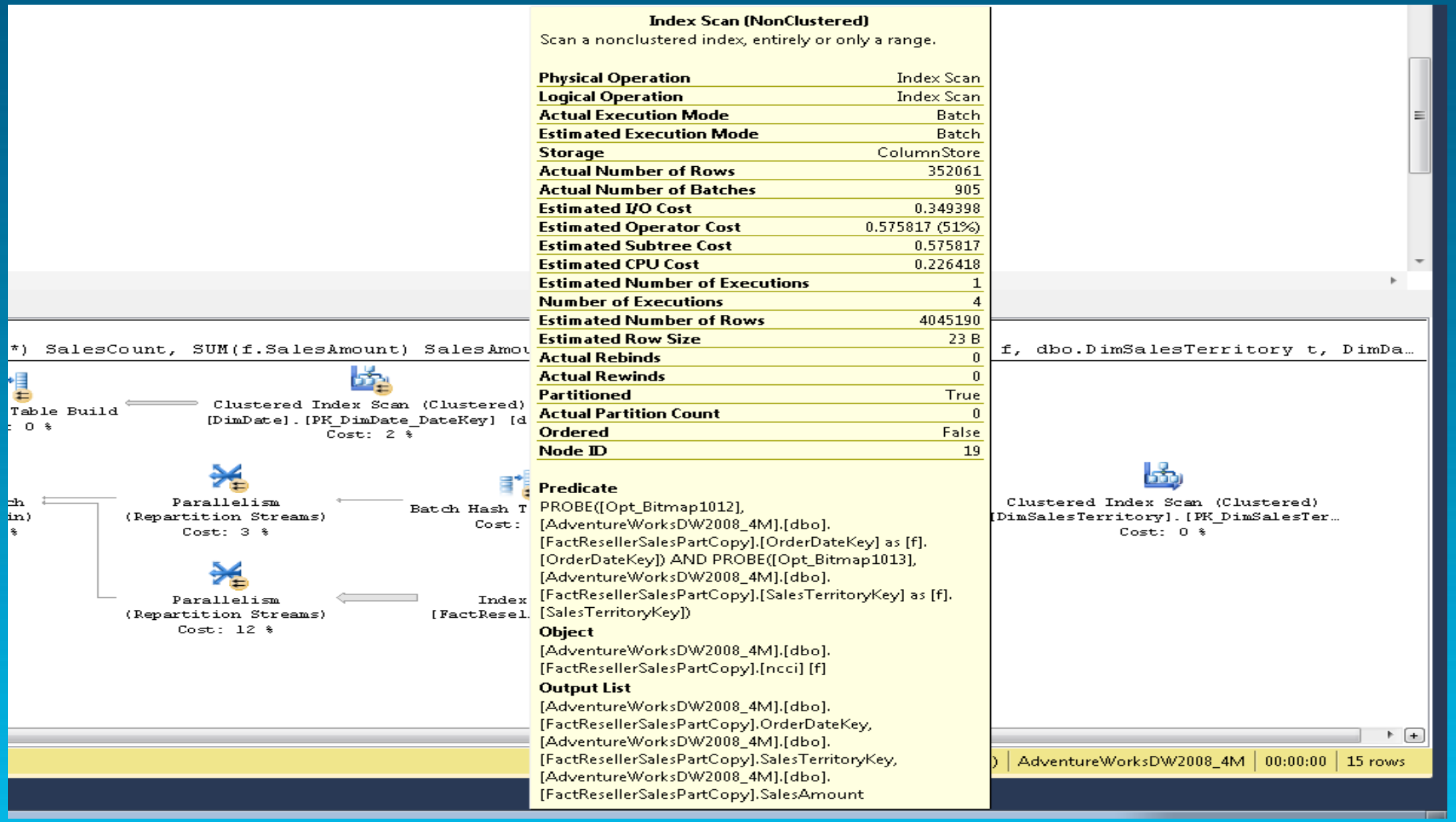


New showplan elements for columnstores and batch processing








Index Scan (NonClustered)
Scan a nonclustered index, entirely or only a range.
Physical Operation Index Scan
Logical Operation Index Scan
Actual Execution Mode Batch
Estimated Execution Mode Batch
Storage ColumnStore
Actual Number of Rows 352061
Actual Number of Batches 905
Estimated I/O Cost 0.349398
Estimated Operator Cost 0.575817 (51%)
Estimated Subtree Cost 0.575817
Estimated CPU Cost 0.226418
Estimated Number of Executions 1
Number of Executions 4
Estimated Number of Rows 4045190
Estimated Row Size 23 B
Actual Rebinds 0
Actual Rewinds 0
Partitioned True
Actual Partition Count 0
Ordered False
Node ID 19
Predicate
PROBE([Opt_Bitmap1012], [AdventureWorksDW2008_4M].[dbo].[FactResellerSalesPartCopy].[OrderDateKey] as [f].[OrderDateKey]) AND PROBE([Opt_Bitmap1013], [AdventureWorksDW2008_4M].[dbo].[FactResellerSalesPartCopy].[SalesTerritoryKey] as [f].[SalesTerritoryKey])
Object
[AdventureWorksDW2008_4M].[dbo].[FactResellerSalesPartCopy].[ncci] [f]
Output List
[AdventureWorksDW2008_4M].[dbo].[FactResellerSalesPartCopy].OrderDateKey, [AdventureWorksDW2008_4M].[dbo].[FactResellerSalesPartCopy].SalesTerritoryKey, [AdventureWorksDW2008_4M].[dbo].[FactResellerSalesPartCopy].SalesAmount
*) SalesCount, SUM(f.SalesAmount) SalesAmou
f, dbo.DimSalesTerritory t, DimDa…
Table Build
: 0 %
Clustered Index Scan (Clustered)
[DimDate].[PK_DimDate_DateKey] [d
Cost: 2 %
Parallelism
(Repartition Streams)
Cost: 3 %
Batch Hash T
Cost:
Parallelism
(Repartition Streams)
Cost: 12 %
Index
[FactResel
Clustered Index Scan (Clustered)
[DimSalesTerritory].[PK_DimSalesTer…
Cost: 0 %
AdventureWorksDW2008_4M
00:00:00
15 rows

Columnstore index catalog tables

```
select name, object_id, index_id, type_desc, number_of_segments from sys.column_store_index_stats;
```

100 %

Results | Messages

	name	object_id	index_id	type_desc	number_of_segments
1	VertiPaq	114099447	6	NONCLUSTERED COLUMNSTORE	4250

```
select hobt_id, column_id, segment_id, row_count, on_disk_size from sys.column_store_segments;
```

100 %

Results | Messages

	hobt_id	column_id	segment_id	row_count	on_disk_size
1	72057594103922688	1	0	1048576	1141968
2	72057594103922688	1	1	1048576	1135464
3	72057594103922688	1	2	1048576	1143176

```
select hobt_id, column_id, dictionary_id, on_disk_size from sys.column_store_dictionaries;
```

100 %

Results | Messages

	hobt_id	column_id	dictionary_id	on_disk_size
1	72057594103922688	1	0	9720
2	72057594103922688	2	0	1204
3	72057594103922688	3	0	1336

Columnstore indexes: interoperability with the rest of SQL Server

- Most things “just work” with columnstore indexes
  - Backup and restore
  - Mirroring
  - Log shipping
  - SSMS
  - Administration, tools

Data type restrictions

- Unsupported types
  - decimal > 18 digits
  - Binary
  - BLOB
  - (n)varchar(max)
  - Uniqueidentifier
  - Date/time types > 8 bytes
  - CLR

Query performance restrictions

- Outer joins
- Unions
- Consider modifying queries to hit “sweet spot”
  - Inner joins
  - Star joins
  - Aggregation

Loading new data into a columnstore index

- Columnstore makes table read-only
- Partition switching allowed
- INSERT, UPDATE, DELETE, and MERGE not allowed
- Two recommended methods for loading data
- Disable, update, rebuild
- Partition switching

Adding data to a table with a columnstore index

- Method 1: Disable the columnstore index
- Disable (or drop) the index
  - ALTER INDEX my_index ON MyTable DISABLE
- Update the table
- Rebuild the columnstore index
  - ALTER INDEX my_index ON MyTable REBUILD

Adding data to a table with a columnstore index

- Method 2: Use Partitioning
- Load new data into a staging table
- Build a columnstore index
  - CREATE NONCLUSTERED COLUMNSTORE INDEX my_index ON StagingT(OrderDate, ProductID, SaleAmount)
- Split empty partition
- Switch the partition into the table
  - ALTER TABLE StagingT SWITCH TO T PARTITION 5

Advanced near-real-time append

- Partitioned table with columnstore index
- Keep copy of data from latest partition in separate row-store table S
- While(True) {
  - Append new data to S for desired period (say 15 minutes)
  - Columnstore index S
  - Swap S with newest partition
  - Drop columnstore index on S
  - Insert rows in S to “catch up” S to current time
  - }

When to build a columnstore index

- Read-mostly workload
- Most updates are appending new data
- Workflow permits partitioning or index drop/rebuild
  - Typically a nightly load window
- Queries often scan & aggregate lots of data

What tables to columnstore index

- Large fact tables
- (Maybe) very large dimension tables

When not to build a columnstore index

- Frequent updates
- Partition switching or rebuilding index doesn’t fit workflow
- Frequent small look up queries
  - B-tree indexes may give better performance
- Your workload does not benefit

Summary: Apollo in a nutshell

Columnstore technology

+

Advanced query processing

Astonishing speedup for DW queries

Thank you!

Microsoft®