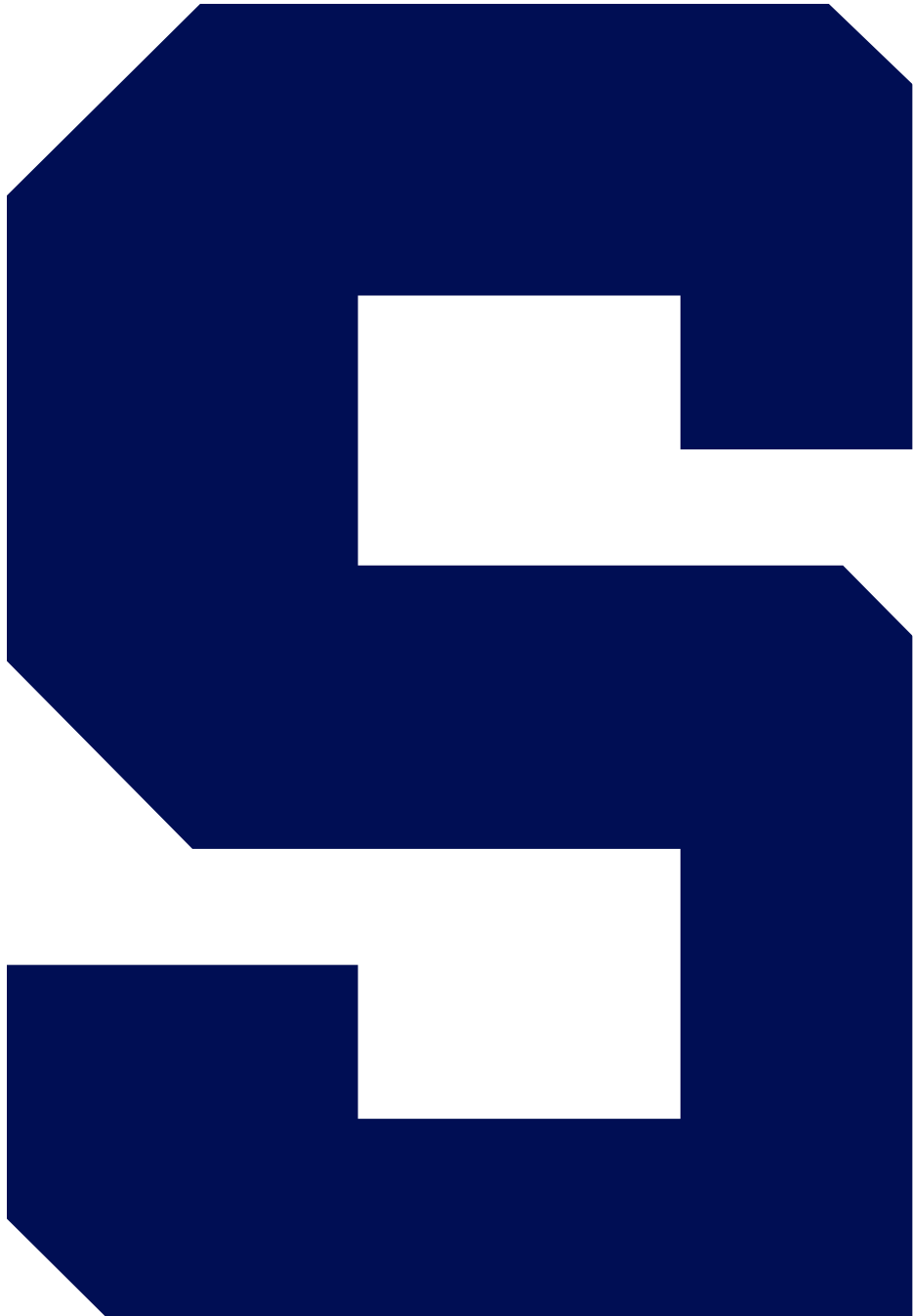
Syracuse University
School of Information Studies



The Physical Model, Performance, and Indexing

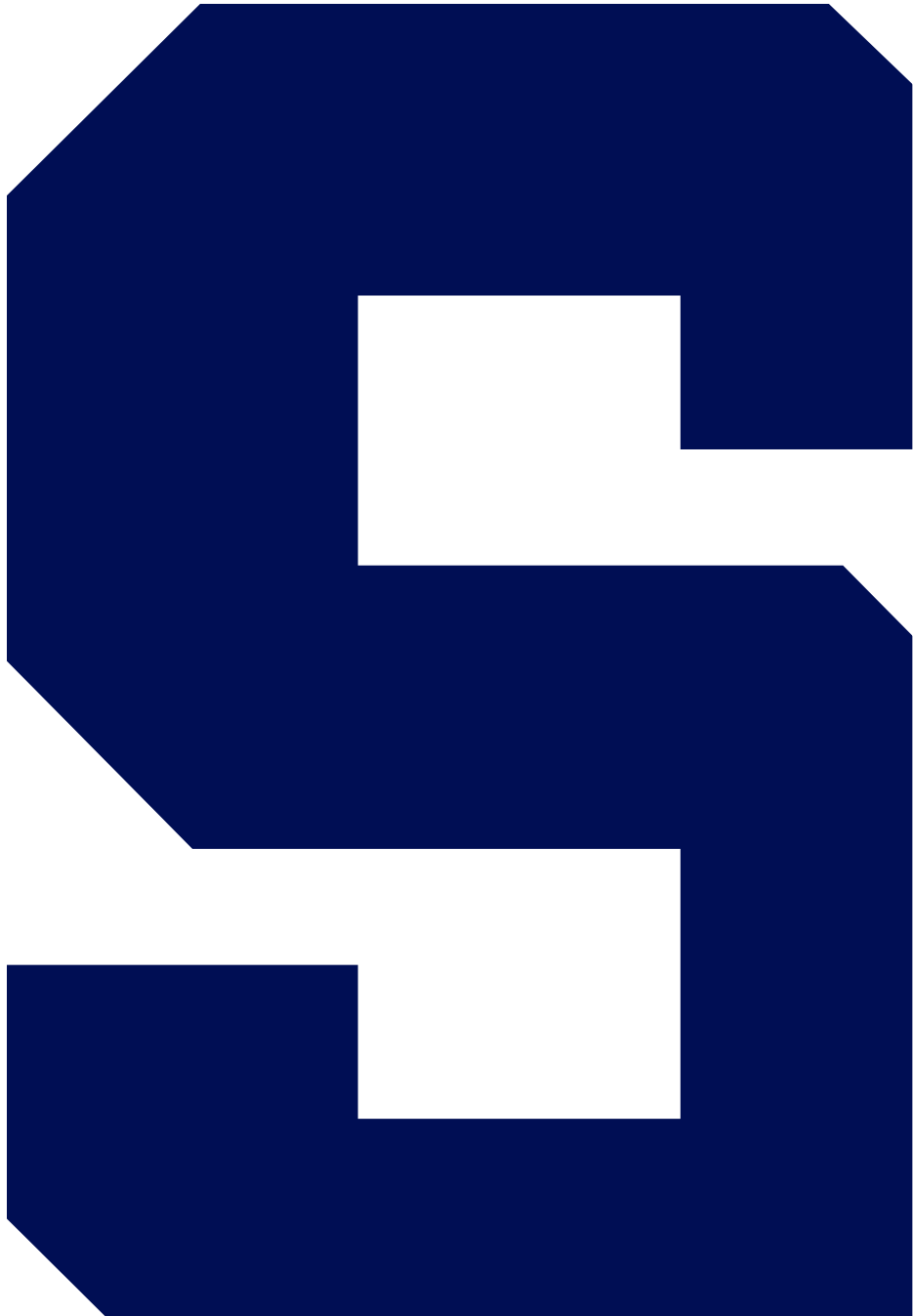
Agenda

- What is the physical data model?
- The details of the SQL Server physical model: pages, files, and filegroups
- Clustered and nonclustered indexes
- Understanding query plans
- Correcting page fragmentation in indexes
- Column store indexes and indexed queries

The Physical Model,
Performance, and Indexing

The End

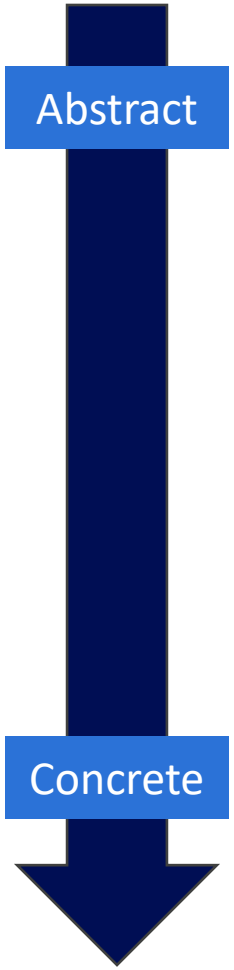
Physical Data Model



Physical Data Model

- Implementation of the logical data model
- Dependent on the selected implementation model (relational, etc.) and the specific DBMS (SQL Server, MySQL, etc.)
- The physical data model is concerned with performance and should consider the DBMS architecture

Recall: Data Models



Conceptual data model

- An abstract representation of the data requirements
- No implementation of the database itself; no implementation model selected

Logical data model

- Mapping of the conceptual model to an implementation model
- No implementation of the database; model selected

Internal data model

- The internalized implementation of the database application
- A DBMS and implementation model selected

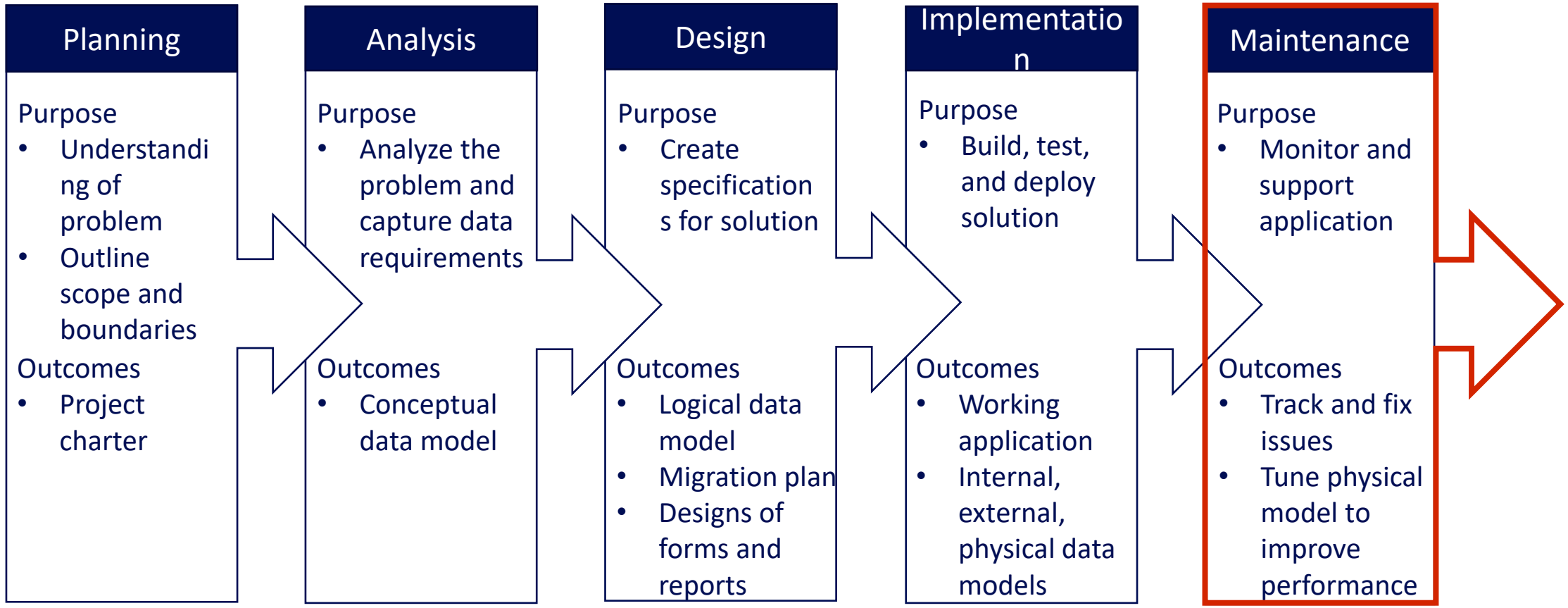
External data model

- User's view of the database application
- DBMS and implementation model selected, internal model implemented

Physical data model

- How the internal/external model is stored by the DBMS and operating system
- All aspects of the database are an implementation

Recall: DBLC


Planning
Purpose
• Understanding of problem
• Outline scope and boundaries
Outcomes
• Project charter
Analysis
Purpose
• Analyze the problem and capture data requirements
Outcomes
• Conceptual data model
Design
Purpose
• Create specifications for solution
Outcomes
• Logical data model
• Migration plan
• Designs of forms and reports
Implementation
Purpose
• Build, test, and deploy solution
Outcomes
• Working application
• Internal, external, physical data models
Maintenance
Purpose
• Monitor and support application
Outcomes
• Track and fix issues
• Tune physical model to improve performance

Physical Database Design

Does it matter?

It does if you care about performance!

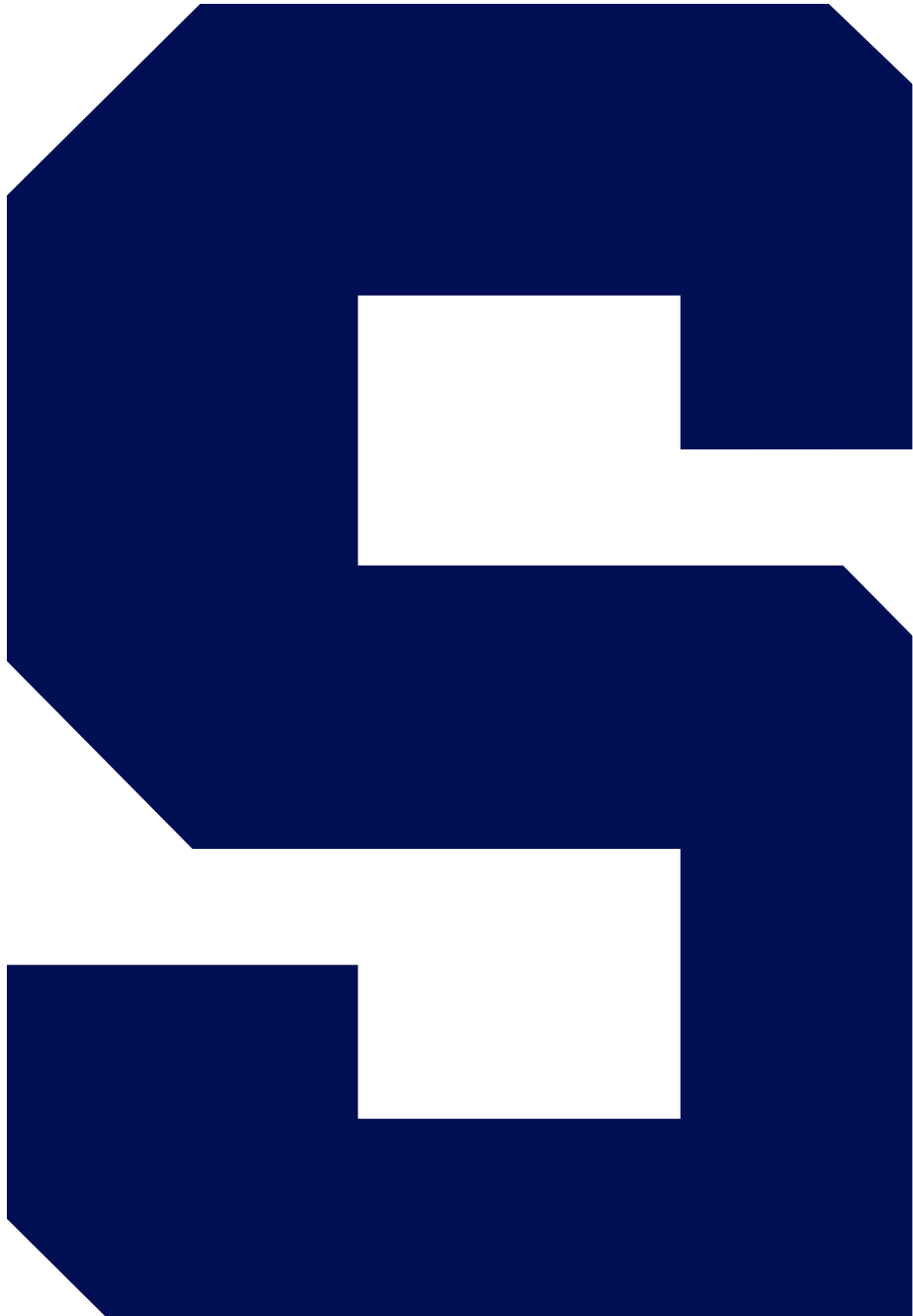
Caveats of Physical Design

- We will explain the physical design of Microsoft SQL Server.
- Other databases have similar concepts.
- Every DBMS has a different means of implementing physical design.
- You can improve performance without knowledge of physical design but will be limited.
- Ultimately, to get the best performance, you must understand the physical design of the database you are using.

Physical Data Model

The End

Syracuse University
School of Information Studies

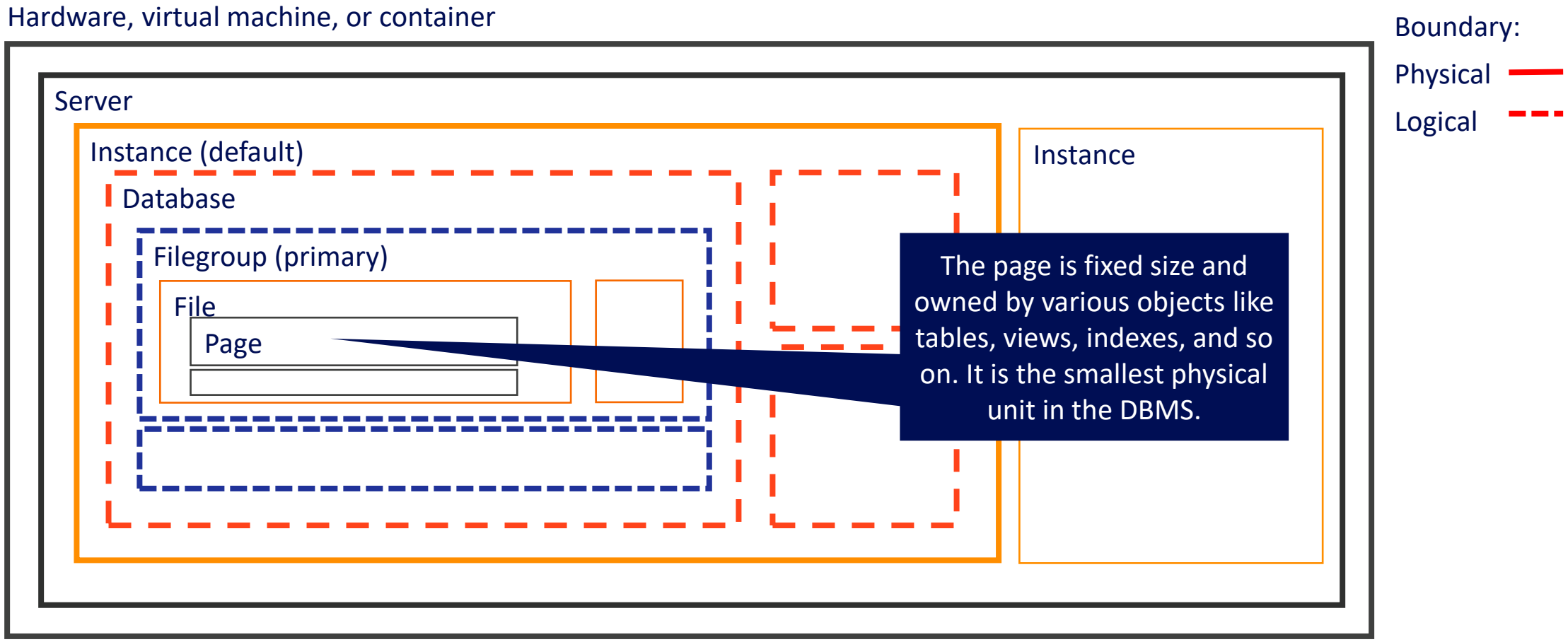


SQL Server
Physical Data Model

The Physical Abstraction of SQL Server

- Server: installed on an operating system or in a container
- Instance: one or more “setups” of SQL Server on the same physical server (test, production, development)
- Database: stored as one or more files
  - *databasename*.mdf → primary file, member of the PRIMARY filegroup
  - Othername.ndf → secondary files used to spread data across physical partitions/disks
  - *databbasename*_log.ldf → transaction log file holds transactions/recovery information
- Filegroup (tablespace): a logical name for one or more physical files; all database objects are written to a filegroup
- Page: set of continuous table rows inside a physical file

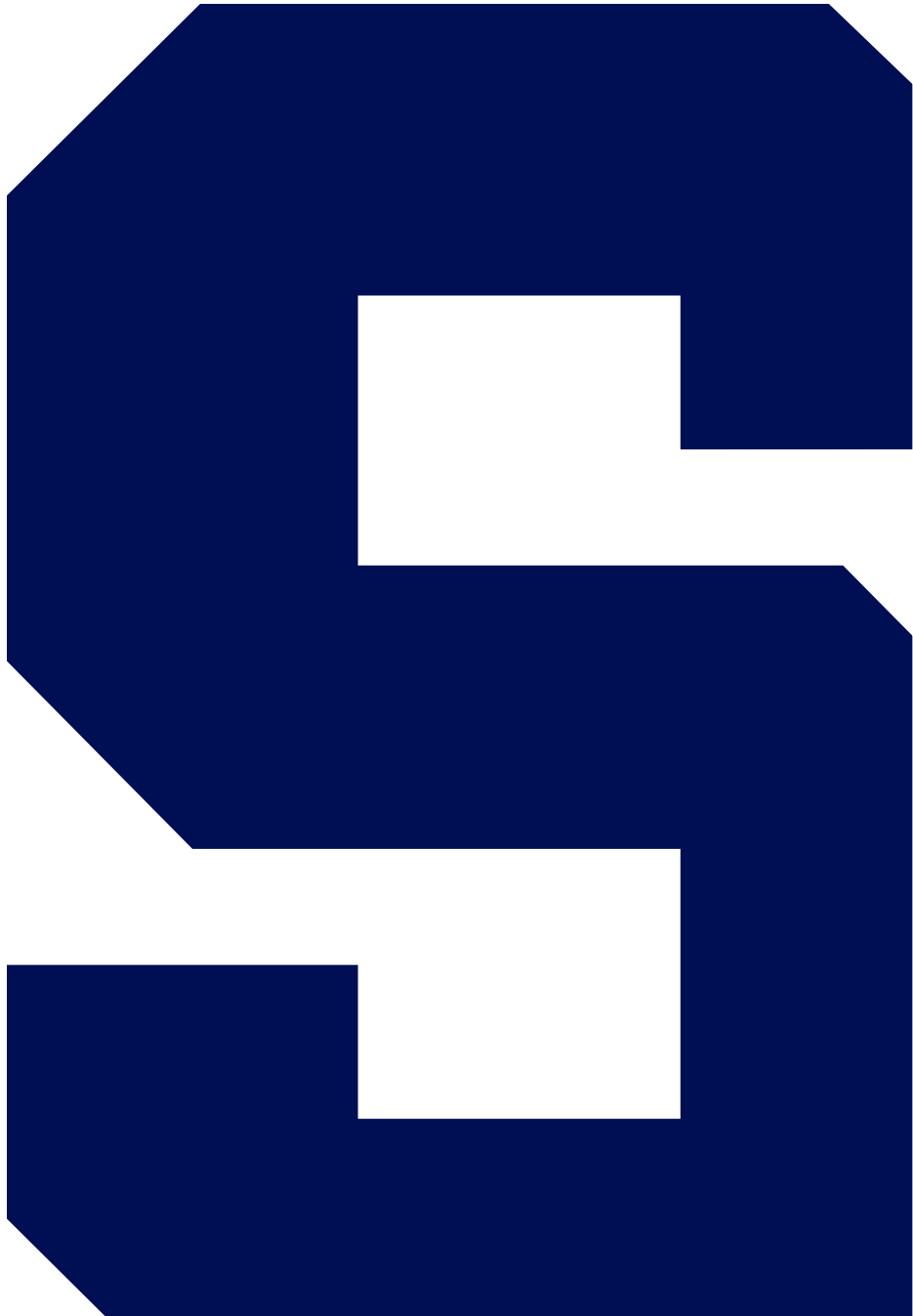
SQL Server Physical Model Visualized



SQL Server and Physical Data Model

The End

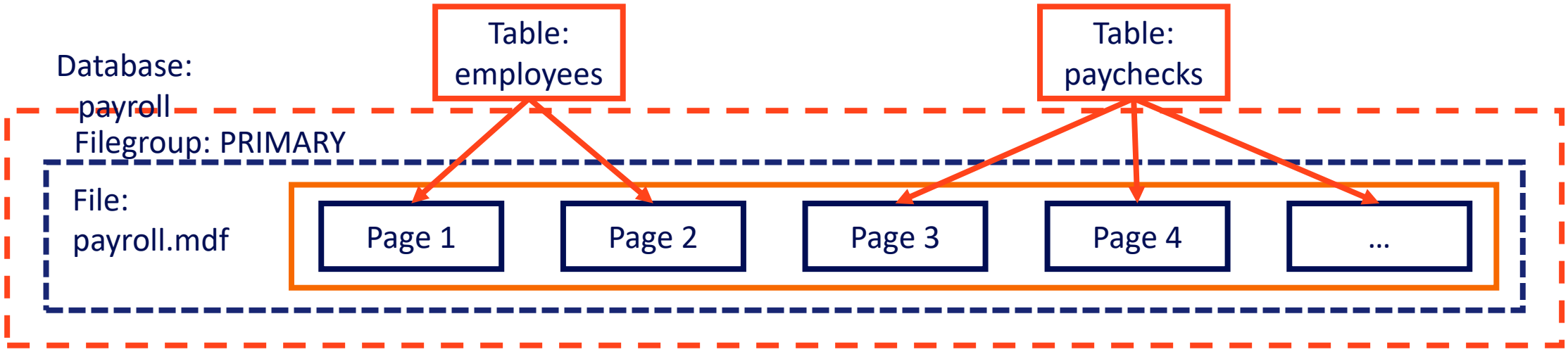
Syracuse University
School of Information Studies



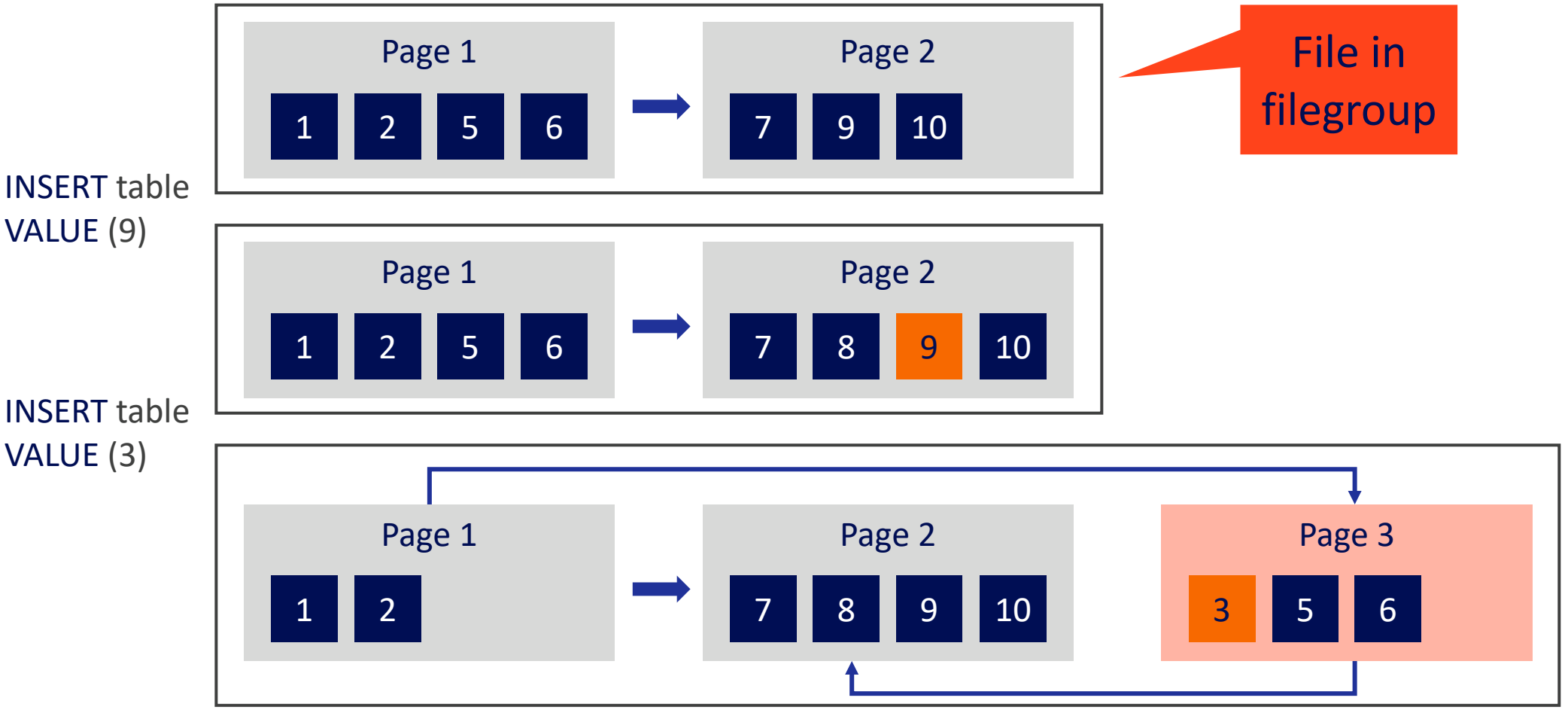
SQL Server Pages

What Is a Page Exactly?

- In the DBMS, rows of data are written to physical blocks inside the file called pages.
- The page size in SQL Server is 8 KB (kilobytes).
- This is the smallest unit of storage in the database. For example, even if you update just one column in a table, an entire page is written.
- The table boundary is the page.



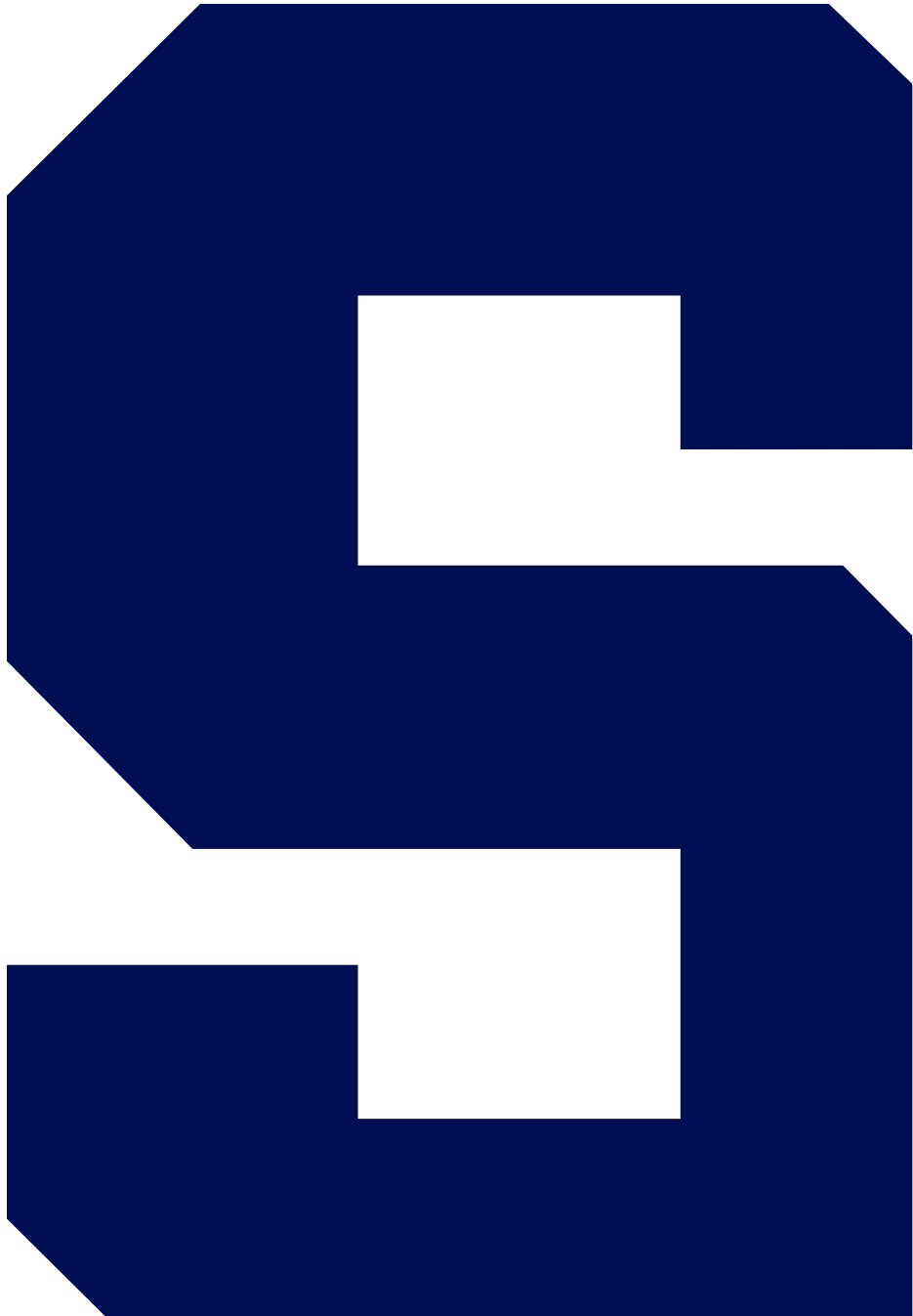
How Data Are Inserted Into Pages



SQL Server Pages

The End

Syracuse University
School of Information Studies



SQL Server
Filegroups and Files

SQL: Filegroups

- Logical storage for database object like tables
- Cannot exist outside a database
- Every SQL Server database as a default filegroup called PRIMARY

```
ALTER DATABASE database_name
    ADD FILEGROUP file_group_name
```

- View the filegroups in a database

```
SELECT *    FROM sys.filegroups
```

SQL: Files in Filegroups

- Filegroups require one or more physical files on disk
- Filegroup is a logical structure; the file is where data are actually stored
- Every SQL Server database as a file database_name.mdf in the default filegroup PRIMARY

```
ALTER DATABASE database_name ADD FILE (
    NAME = logical_name,
    FILENAME = path_to_file
) TO FILEGROUP file_group_name
```

SQL: File/Filegroup Inspection

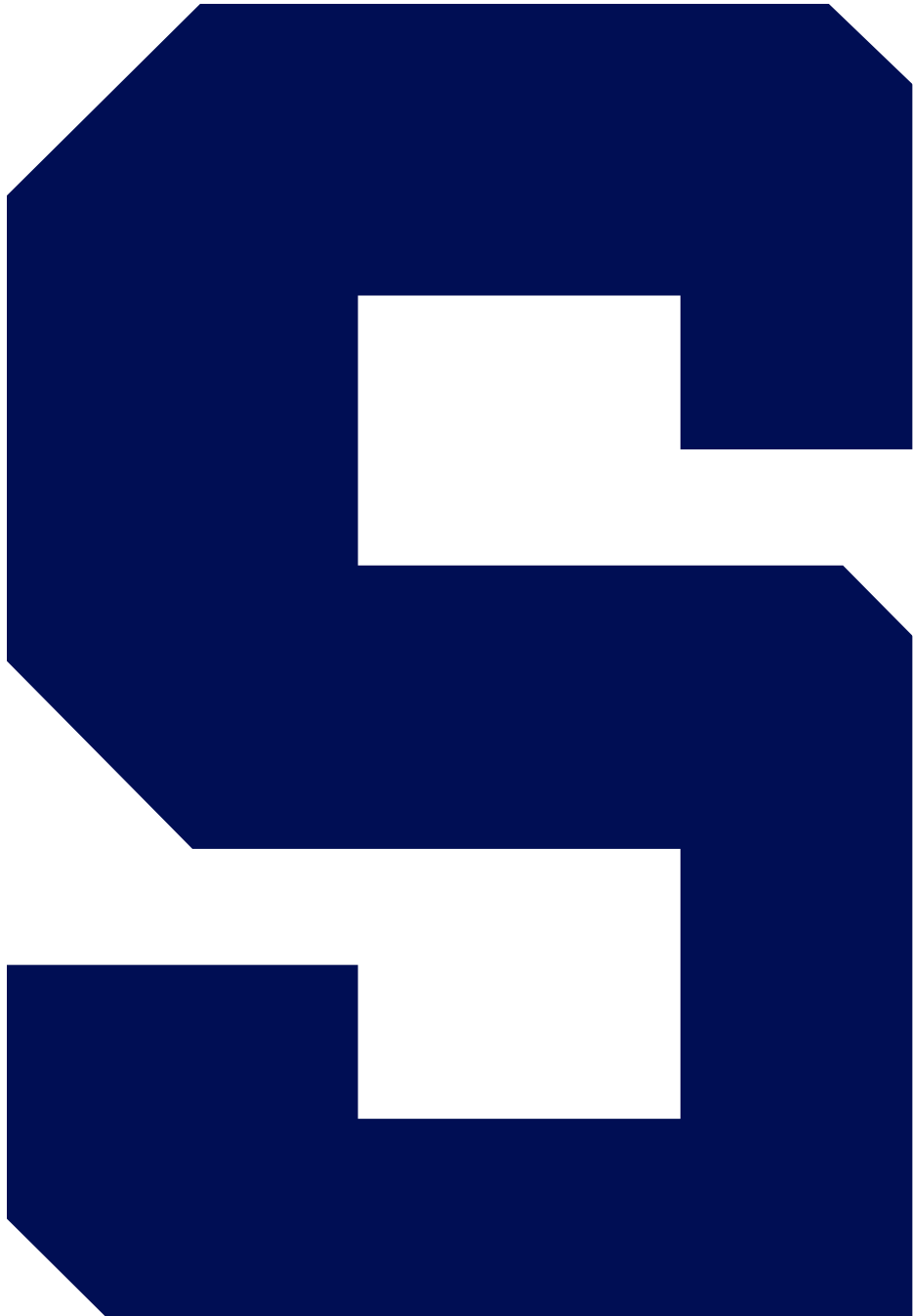
- View the filegroups in a database.
  `SELECT * FROM sys.filegroups`
- View the filegroups in a database.
  `SELECT * FROM sys.database_files`
- Show the objects in a filegroup. This is tricky.

```
SELECT o.[name] as object_name,  i.[name] as index_name,  f.[name] as filegroup_name
    FROM sys.indexes i
        JOIN sys.filegroups f ON i.data_space_id = f.data_space_id
        JOIN sys.all_objects o ON i.[object_id] = o.[object_id]
    WHERE i.data_space_id = f.data_space_id AND o.type = 'U' -- User Created Tables
```

SQL Server Filegroups and Files

The End

Syracuse University
School of Information Studies



Demo

Database, Filegroups, and Files

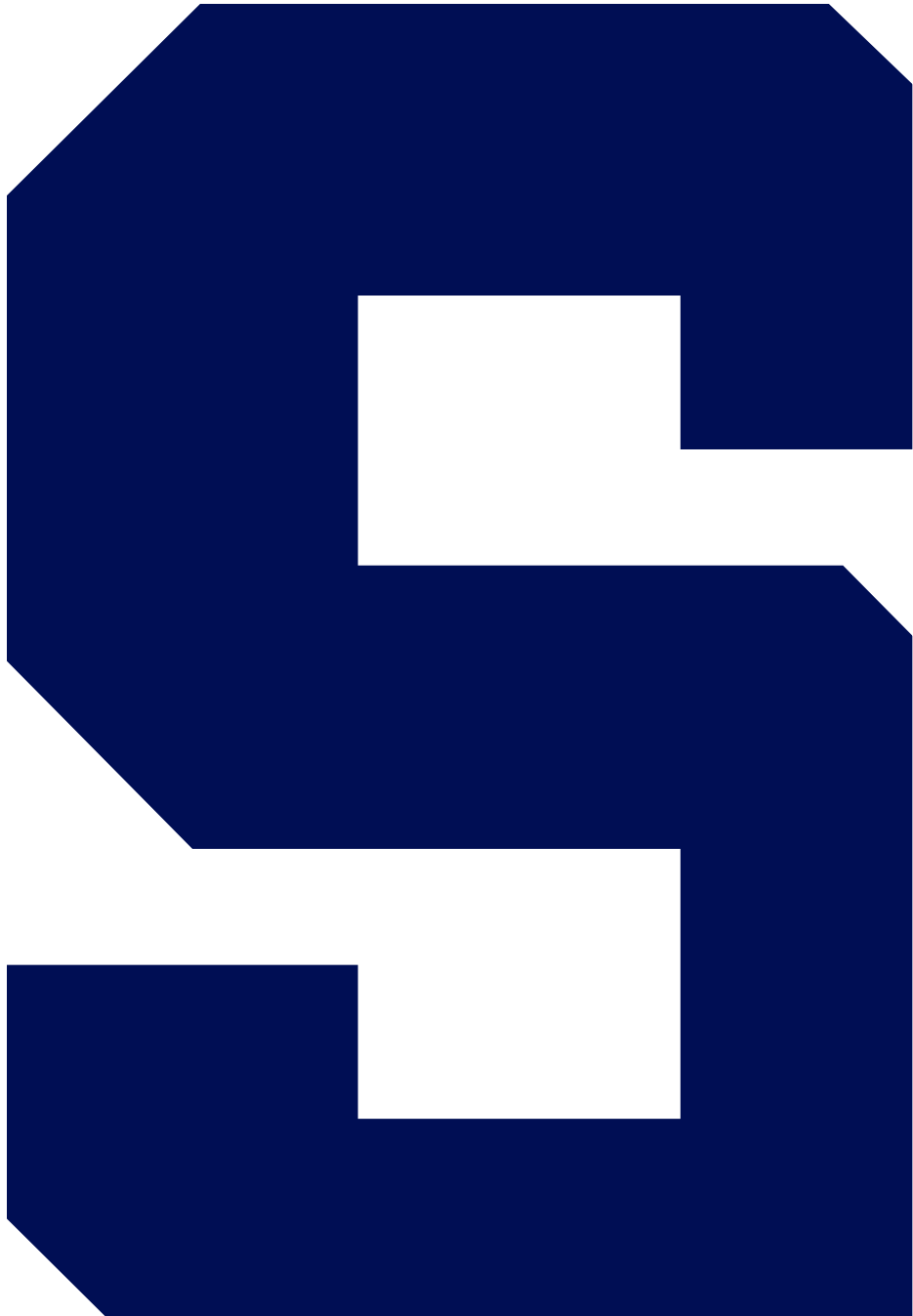
Demo: Database, Filegroups, and Files

- We will use the Azure Data Studio application
- We will use the demo database
- Add filegroup to demo
- Show filegroups
- Create table but cannot insert, as there is no file
- Add file to filegroup on demo
- Insert and show it now works
- Show files in filegroups, objects in filegroup
- Move object to another filegroup
- Show physical files on server instance

Demo: Database, Filegroups, and Files

The End

Syracuse University
School of Information Studies



Clustered Indexes

Database Index

- Improves the searchability of our data, at the expense of maintaining multiple versions of it
- Just like the index in the back of a book, takes us to a page where that word occurs
- The database index takes us to the database page where the data for which we're searching can be found
- Scan traverses linearly, a page at a time
- Seek jumps to the page with the content

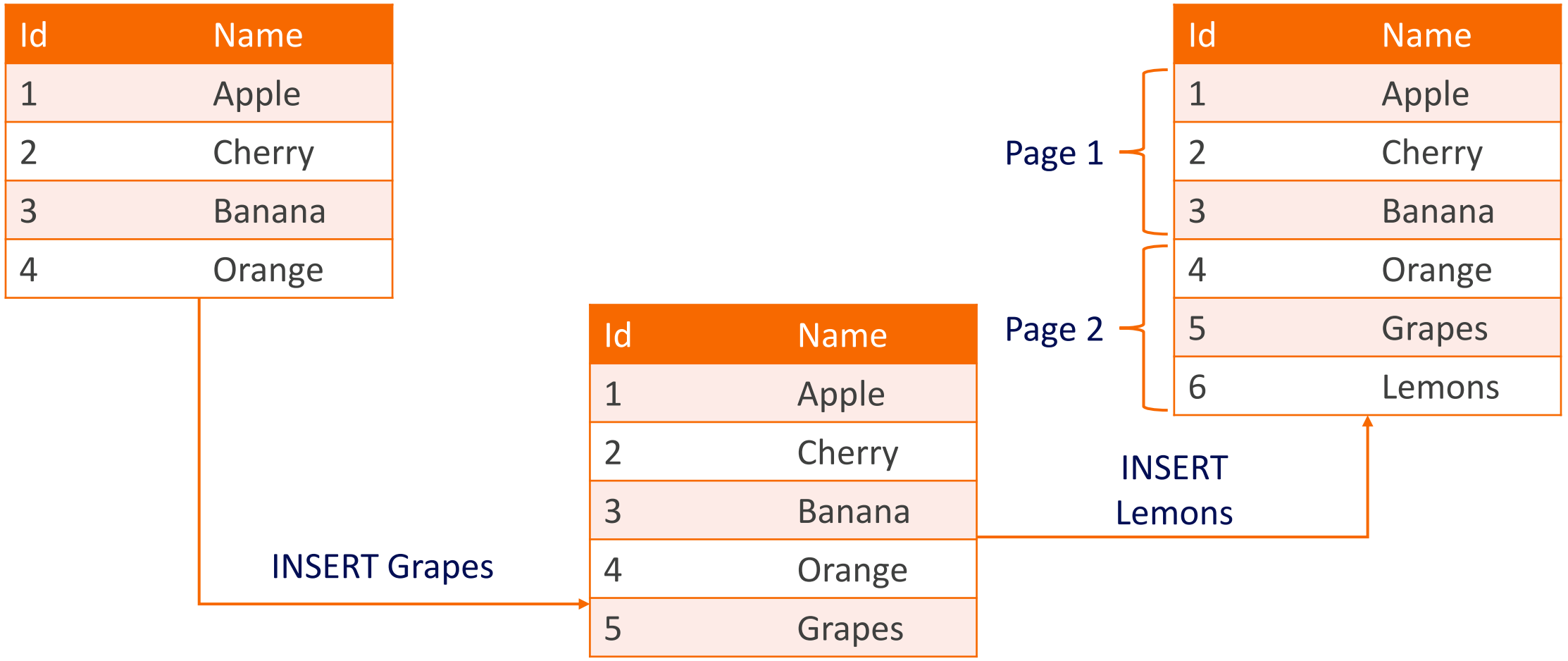
Clustered Index

- Sorts and stores the data in the order of the key values
- There can only be one clustered index per table since rows can only be stored in that order
- By default, the primary key contains a clustered index, but this is not a requirement
- The pages are organized by the clustered index

Primary Key as a Clustered Index

- This is why auto-incrementing values are good for clustered indexes—they avoid page fragmentation
  - Int identity
  - Date/time stamp
- Ideal key/clustered index
  - Narrow: not many bytes in size
  - Unique
  - Static: never changing
  - Ever-increasing

Clustered Index Visualized



CREATE TABLE Revisited

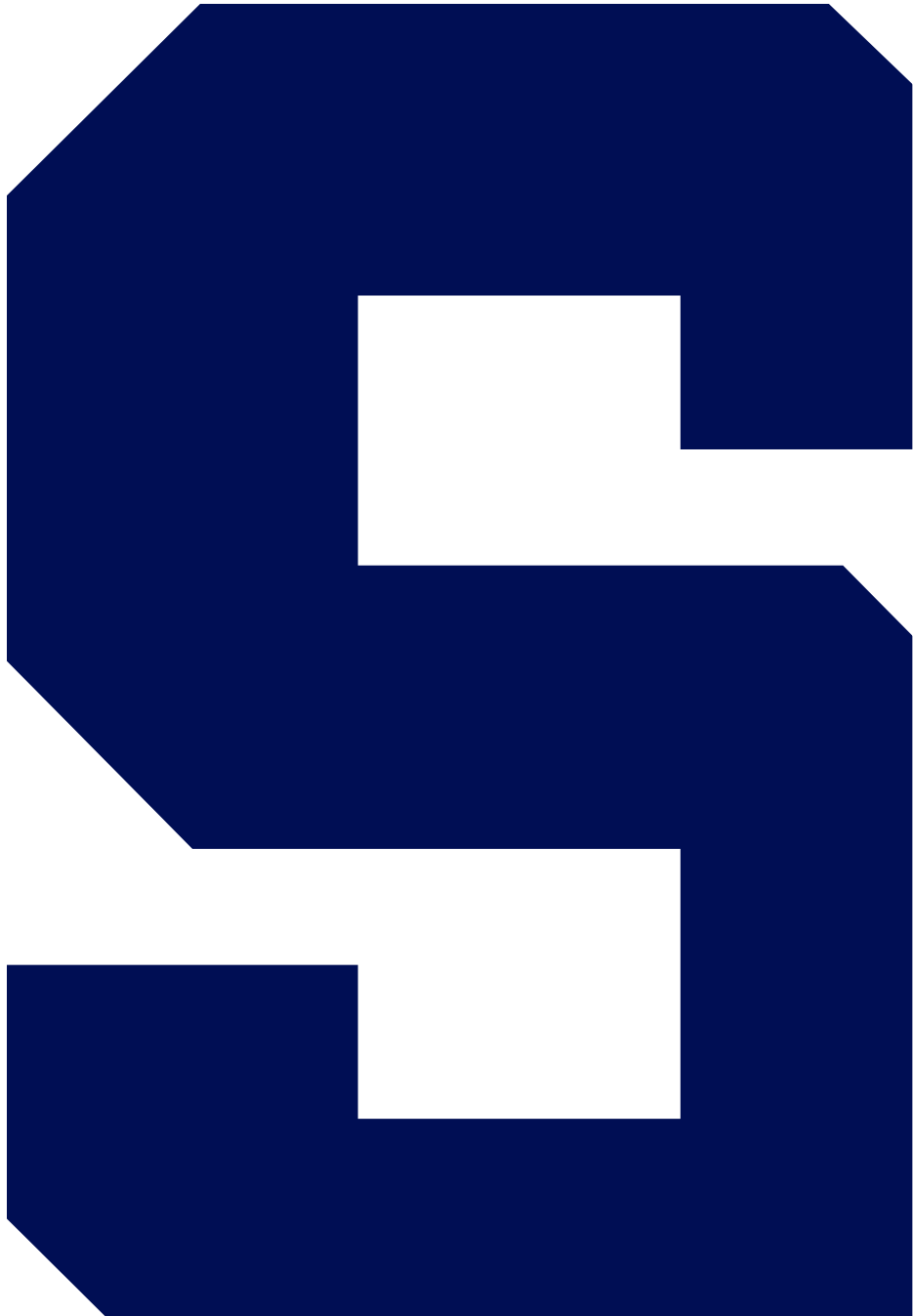
```
CREATE TABLE table_name (
    column,
    […],
    CONSTRAINT pk_name PRIMARY KEY CLUSTERED| NONCLUSTERED
        (columns) [ON filegroup_name]
) [ON filegroup_name]
```

- PK can be clustered or nonclustered.
- Separate filegroups can be specified for the table and the PK.
- A table can only have one clustered index!
- Nonclustered PKs make sense for natural keys.

Indexes

The End

Syracuse University
School of Information Studies



Demo

Tables and Primary Keys Revisited

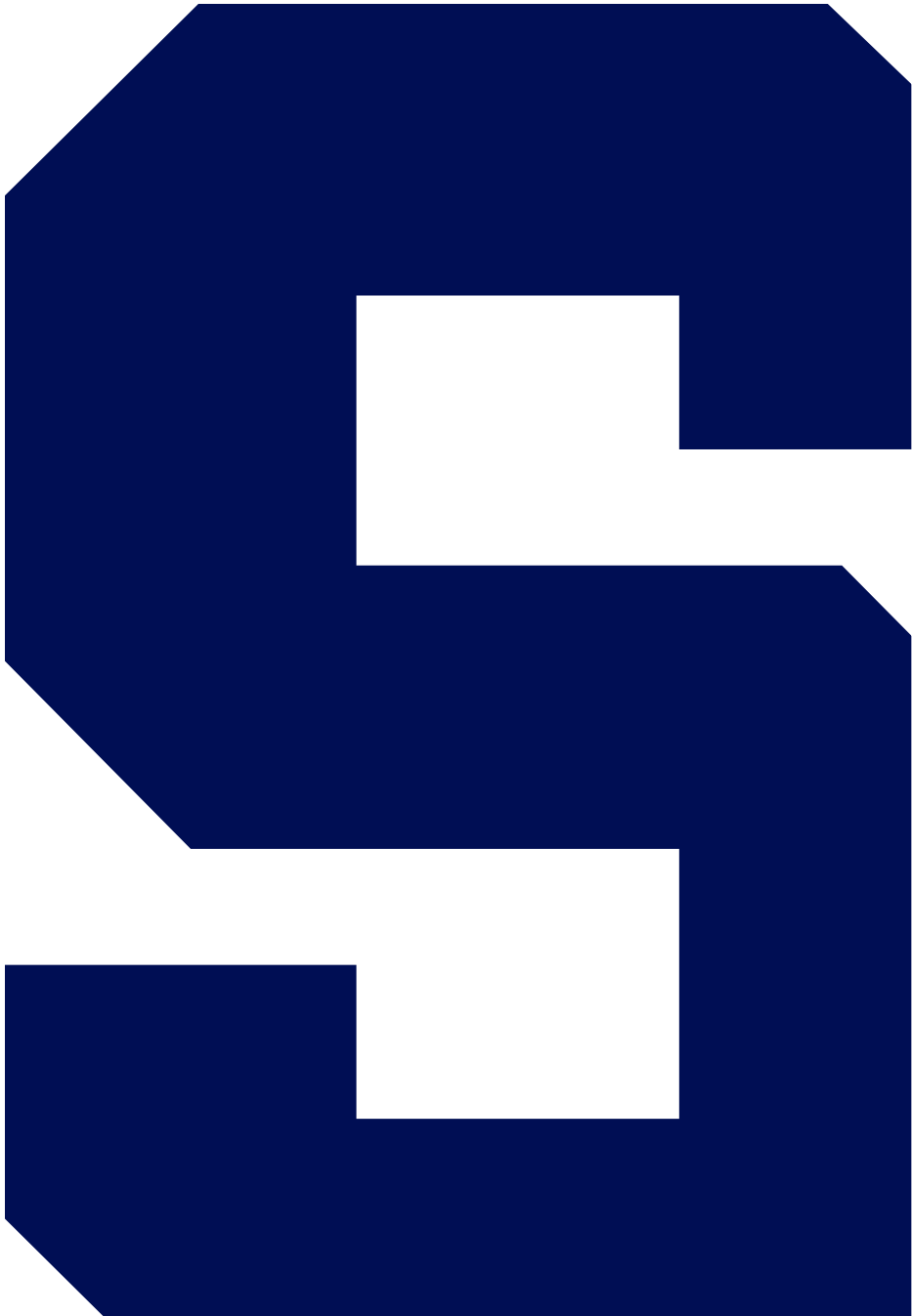
Demo: Tables and Primary Keys Revisited

- We will use the Azure Data Studio application
- We will use the demo database
- Primary key with nonclustered index
- Only can have one clustered index on a table
- Natural keys should be nonclustered
- Create table clustered/nonclustered
- You can store the index in the same filegroup or a different filegroup from the table

Demo: Tables and Primary Keys Revisited

The End

Nonclustered Indexes



Nonclustered Index

- Secondary indexes compared with the primary clustered index
- Use to improve the performance of queries
- You can create multiple nonclustered indexes on your table
- The cost is extra space and extra inserts
- Every index on a table must be updated when the data in a table are updated

Create Index

```
CREATE [UNIQUE] INDEX index_name
      ON table_name (column,[…])
      [INCLUDE (column, […])]
      [ON filegroup_name]
```

- The columns next to the table are the index keys.
- The index can be made unique—a constraint.
- The columns in the INCLUDE clause are not part of the index key, but included in the index.
- When these columns are in the projection of the SELECT statement, you get an index seek.

Nonclustered Index Visualized

Id	Name	Type
1	Apple	Fruit
2	Carrot	Vegetable
3	Banana	Fruit
4	Orange	Fruit
5	Celery	Vegetable

Type	Name	Id
Fruit	Apple	1
	Banana	3
	Orange	5
Vegetable	Carrot	2
	Celery	4


```
CREATE INDEX ix_produce_type
      ON produce (Type) INCLUDE (Name)
```

Faster Reads

Id	Name	Type
1	Apple	Fruit
2	Carrot	Vegetable
3	Banana	Fruit
4	Orange	Fruit
5	Celery	Vegetable

Type	Name	Id
Fruit	Apple	1
	Banana	3
	Orange	5
Vegetable	Carrot	2
	Celery	4

```
SELECT Name FROM produce
WHERE Type='Fruit'
```

The index retrieves 1 row vs. scanning 5 rows that should be included in the query.

Slower Writes

Id	Name	Type
1	Apple	Fruit
2	Carrot	Vegetable
3	Banana	Fruit
4	Orange	Fruit
5	Celery	Vegetable
6	Tomato	Fruit

Type	Name	Id
Fruit	Apple	1
	Banana	3
	Orange	5
	Tomato	6
Vegetable	Carrot	2
	Celery	4

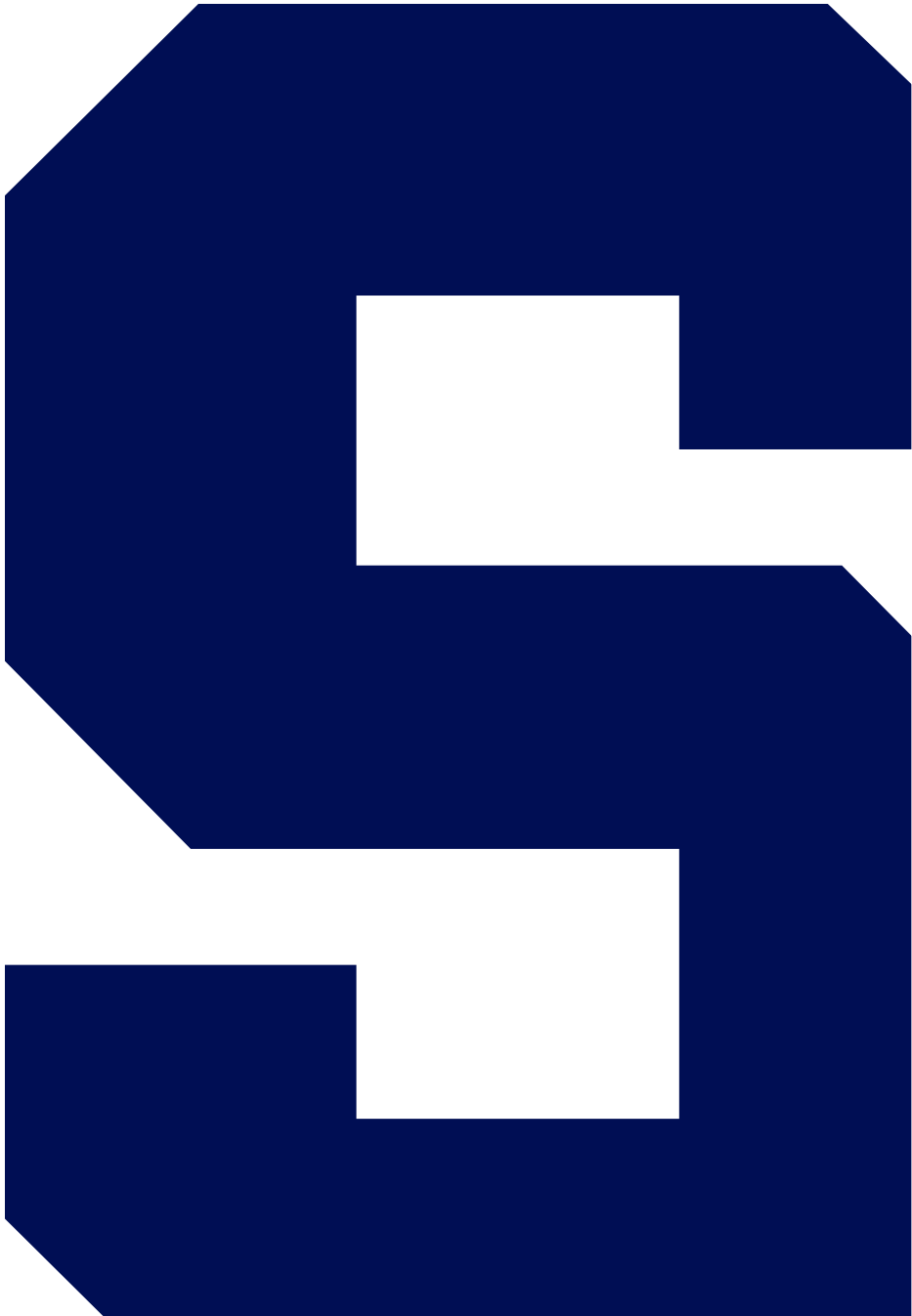
```
INSERT INTO produce
VALUES (6, 'Tomato','Fruit')
```

Every table write also forces a write to the index.

Nonclustered Indexes

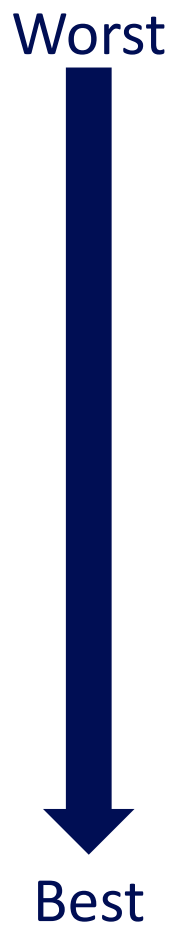
The End

Query Tuning



Scans and Seeks: Finding a Row of Data

- Table scan: search for—row by row—data in each column; worst-case scenario
- Clustered index scan: use the clustered index (primary key, usually) to search row by row
- Clustered index seek: jumps to the correct page using the clustered index
- Index scan: search the index row by row—better than scanning the clustered index, as it typically has fewer rows
- Index seek: use the index to jump to the correct page—Nirvana!



Query Plan Tool Shows Data Access

```
1   select student_firstname, student_lastname, student_gpa
2       from students
3       where student_year_name = 'Freshman'
4
```

Results Messages **Query Plan** Top Operations

Query 1

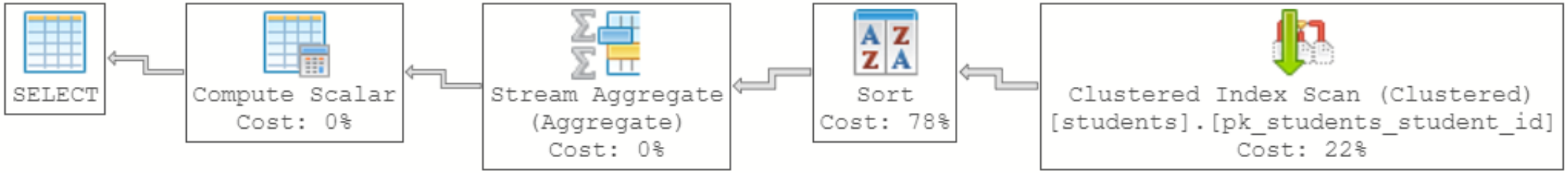
elect student_firstname, student_lastname, student_gpa from students where student_year_name = 'Freshman'

Still an Index Scan but Now Needs a Sort

```
6   select student_year_name, avg(student_gpa)
7       from students
8       group by student_year_name
9
```

Results Messages Query Plan Top Operations

Query 1
select student_year_name, avg(student_gpa) from students group by student_year_name



How to Cover Your Query With an Index

- Index key should be the column(s) in the:
  - WHERE
  - GROUP BY or
  - HAVING classes
- Included index columns are columns from the projection
  - SELECT line

```
drop index if exists  ix_students_by_year_name on students
go
create NONCLUSTERED index ix_students_by_year_name
    on students (student_year_name)
    include (student_firstname, student_lastname, student_gpa)
```

Query Plan: Before Index

```
1   select student_firstname, student_lastname, student_gpa
2       from students
3       where student_year_name = 'Freshman'
4
```

Results Messages Query Plan Top Operations

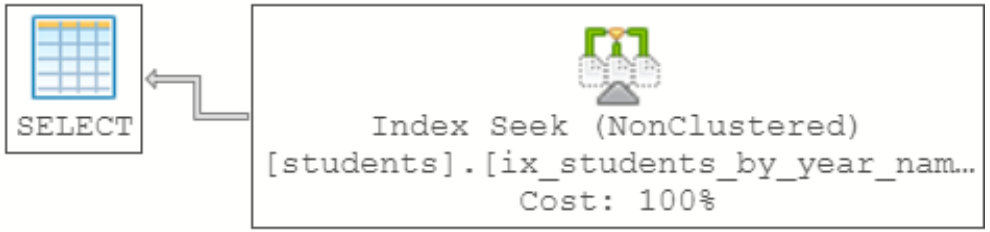
Query 1
elect student_firstname, student_lastname, student_gpa from students where student_year_name = 'Freshman'

Query Plan: After Index

```
1   select student_firstname, student_lastname, student_gpa
2       from students
3       where student_year_name = 'Freshman'
4
```

Results Messages Query Plan Top Operations

Query 1
elect student_firstname, student_lastname, student_gpa from students where student_year_name = 'Freshman'

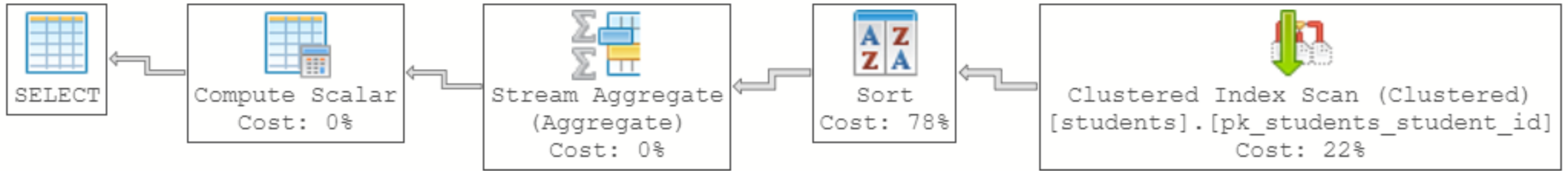


Query Plan: Before Index

```
6   select student_year_name, avg(student_gpa)
7       from students
8       group by student_year_name
9
```

Results Messages Query Plan Top Operations

Query 1
select student_year_name, avg(student_gpa) from students group by student_year_name

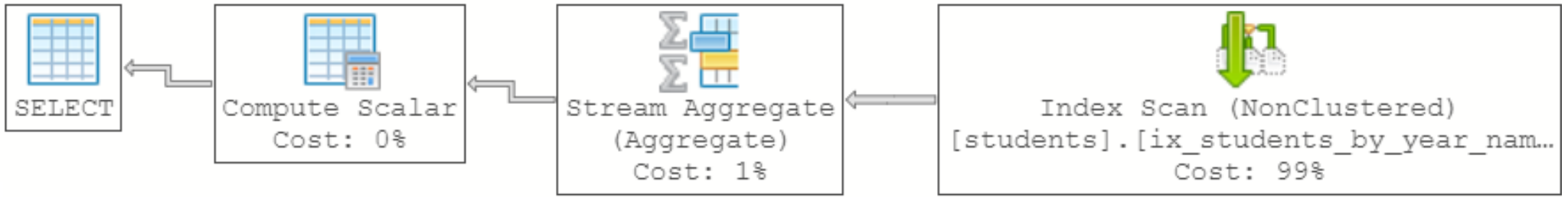


Query Plan: After Index

```
6   select student_year_name, avg(student_gpa)
7       from students
8       group by student_year_name
9
```

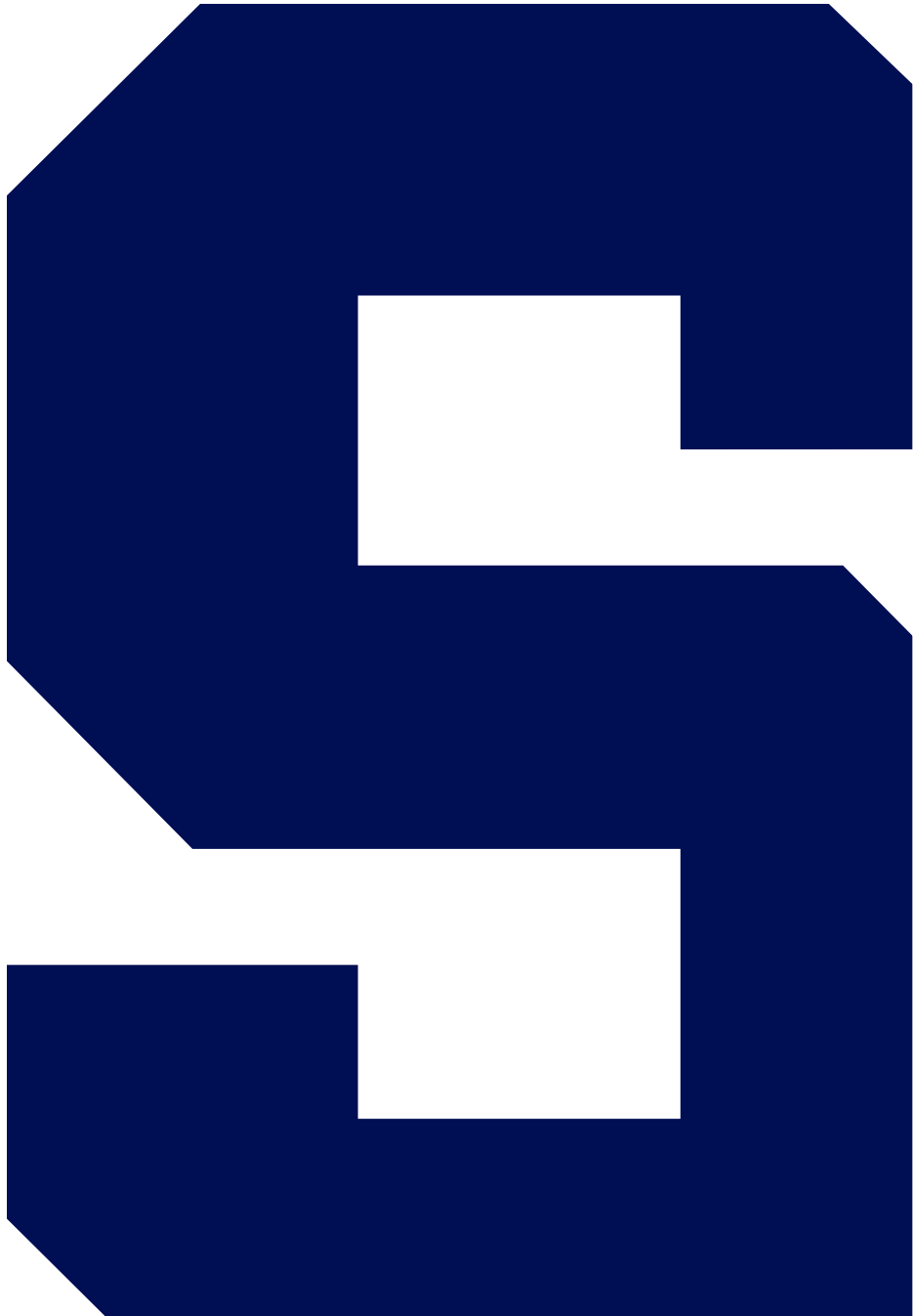
Results Messages Query Plan Top Operations

Query 1
select student_year_name, avg(student_gpa) from students group by student_year_name



Query Tuning

The End



Demo

Understanding Query Plans

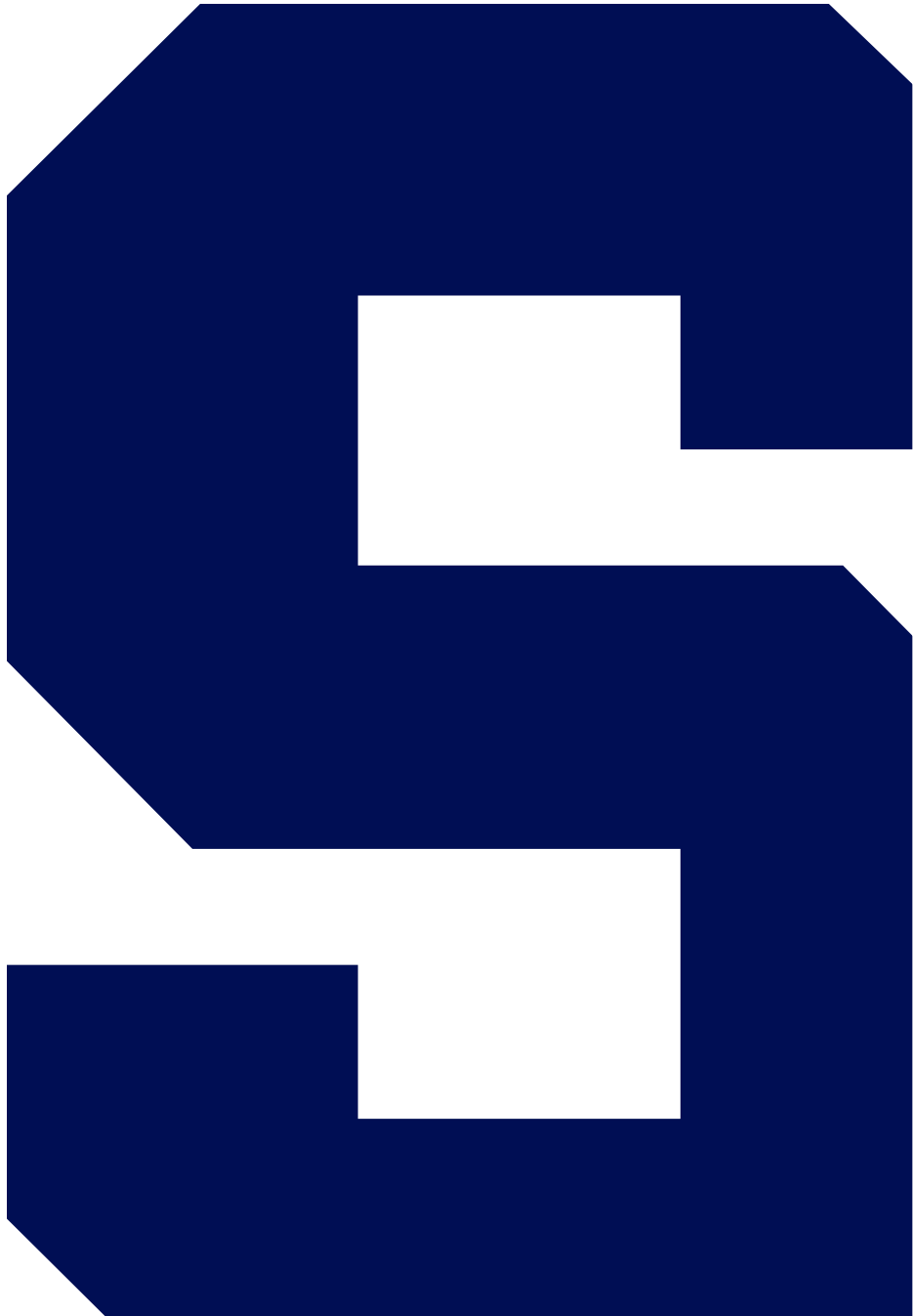
Demo: Understanding Query Plans

- We will use the Azure Data Studio application
- We will use the tinyu database
- Example of how to read query plans
- How window functions affects the query
- An adjustment to the window function improves performance
- Remove the sort operation with an index

Demo: Understanding Query Plans

The End

Syracuse University
School of Information Studies



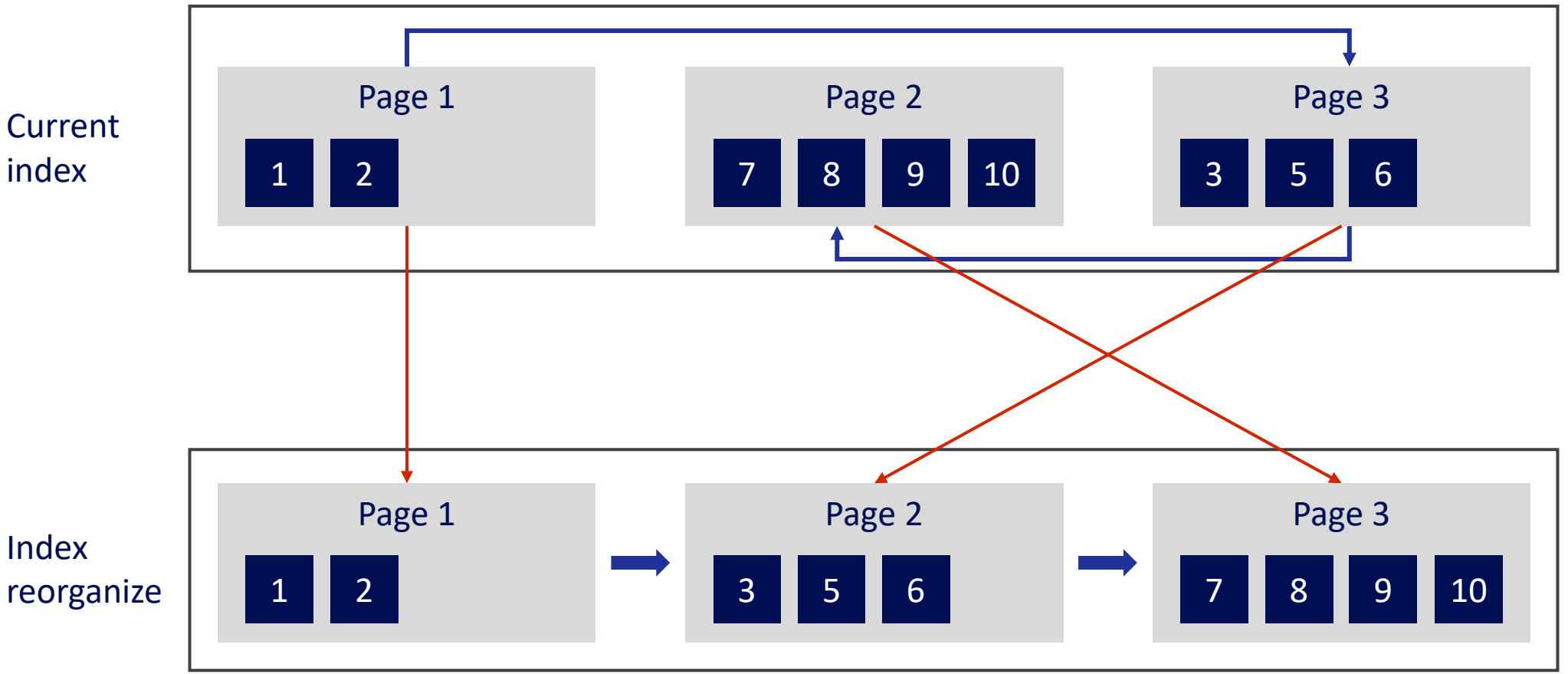
Index Fragmentation

Rebuilding Indexes

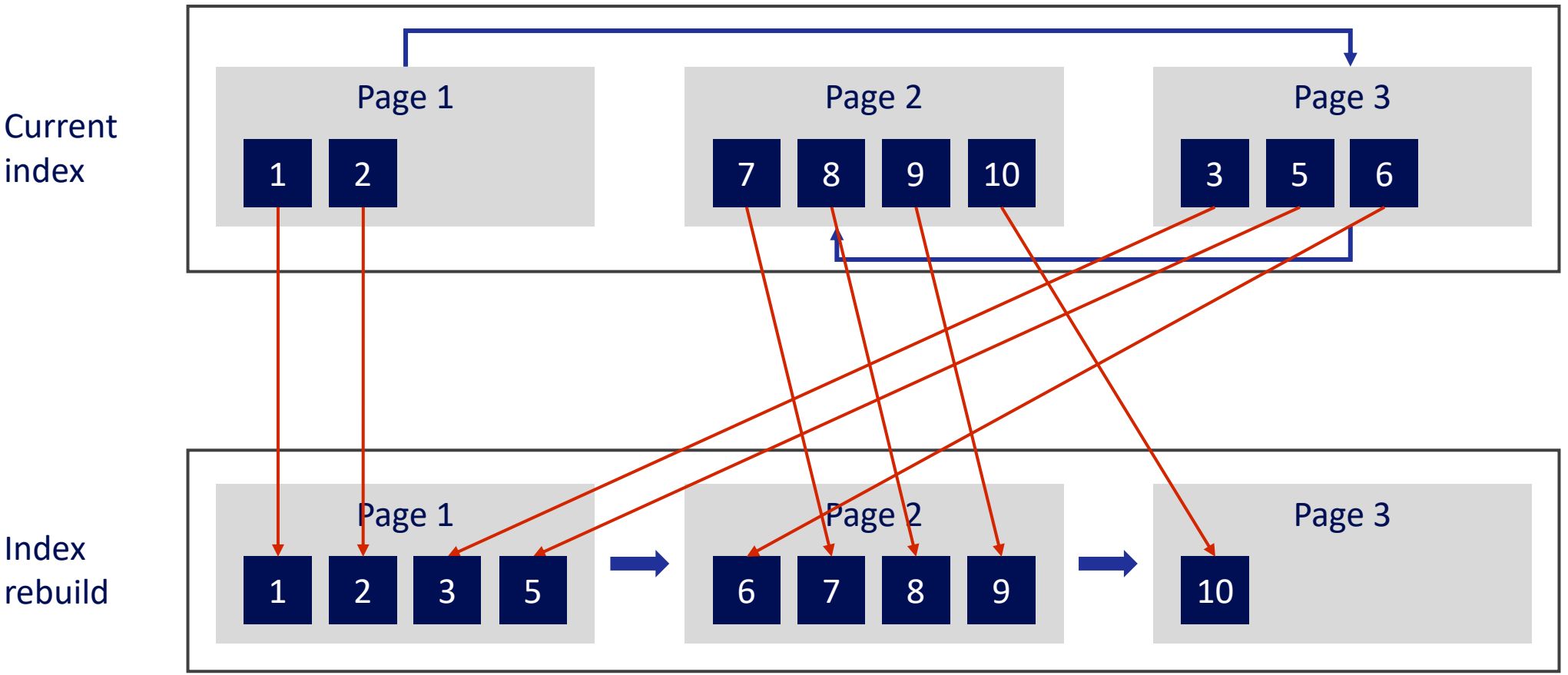
- As data are inserted, updated, and deleted into your tables, your indexes will no longer be optimal
- The pages will not be stored efficiently, and your indexes become fragmented
- Reorganize: move the pages around
- Rebuild: create the index from scratch, fill factor is percent of each page to fill

```
ALTER INDEX index_name ON table REORGANIZE
ALTER INDEX index_name ON table REBUILD
     [WITH FILLFACTOR=n)]
```

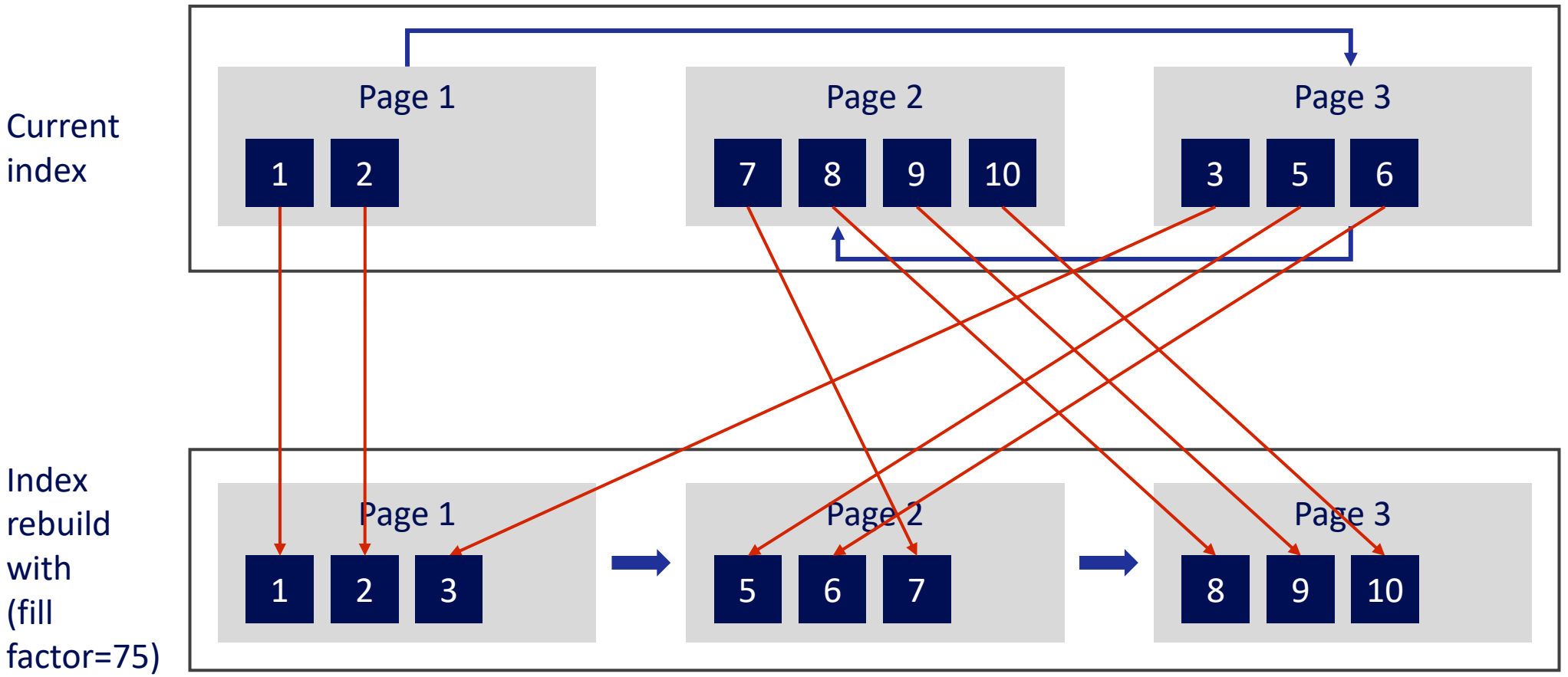
Index Page Fragmentation: Reorganize



Index Page Fragmentation: Rebuild

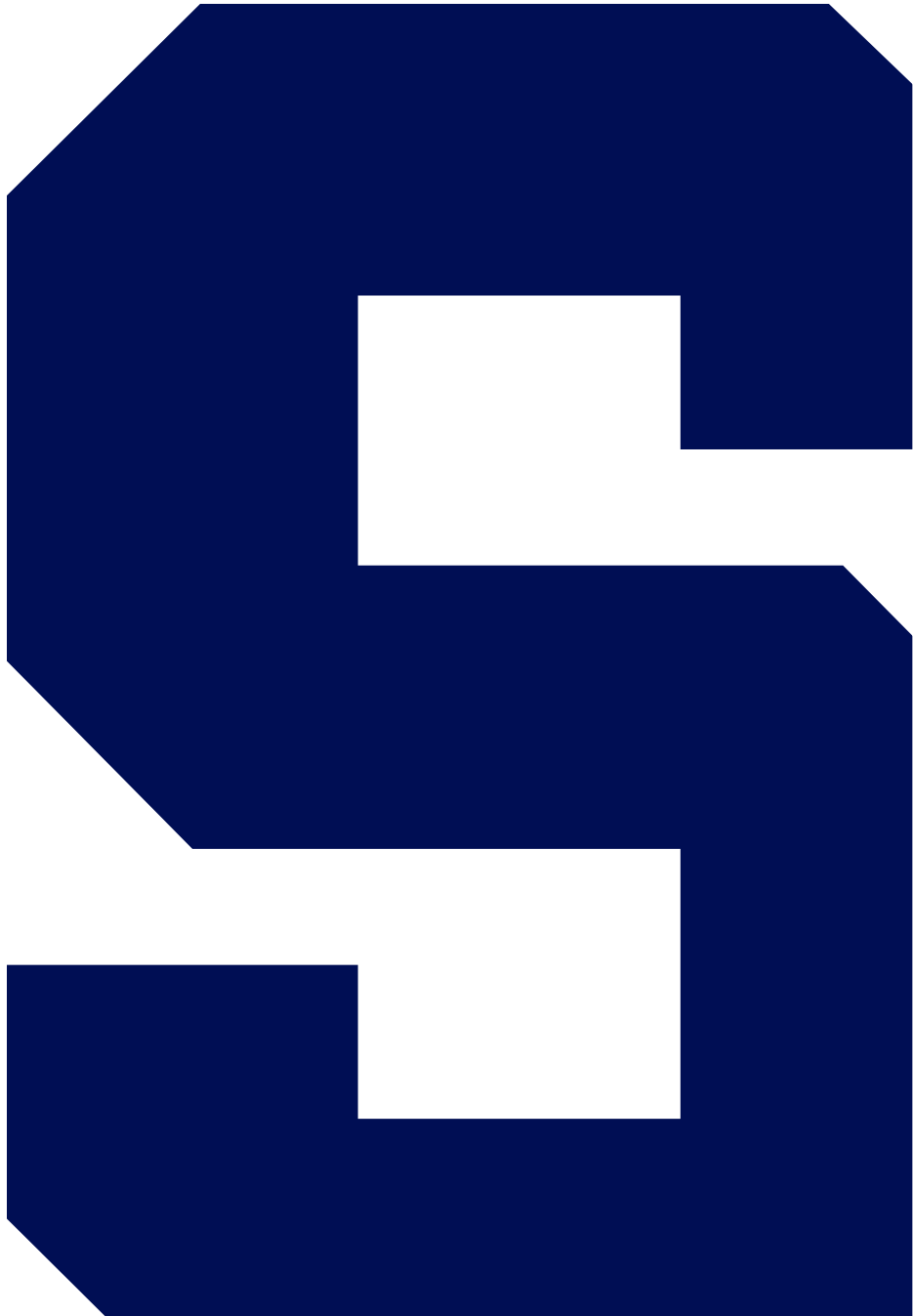


Index Page Fragmentation: Rebuild Fill Factor



Index Fragmentation

The End



Demo

Index Statistics

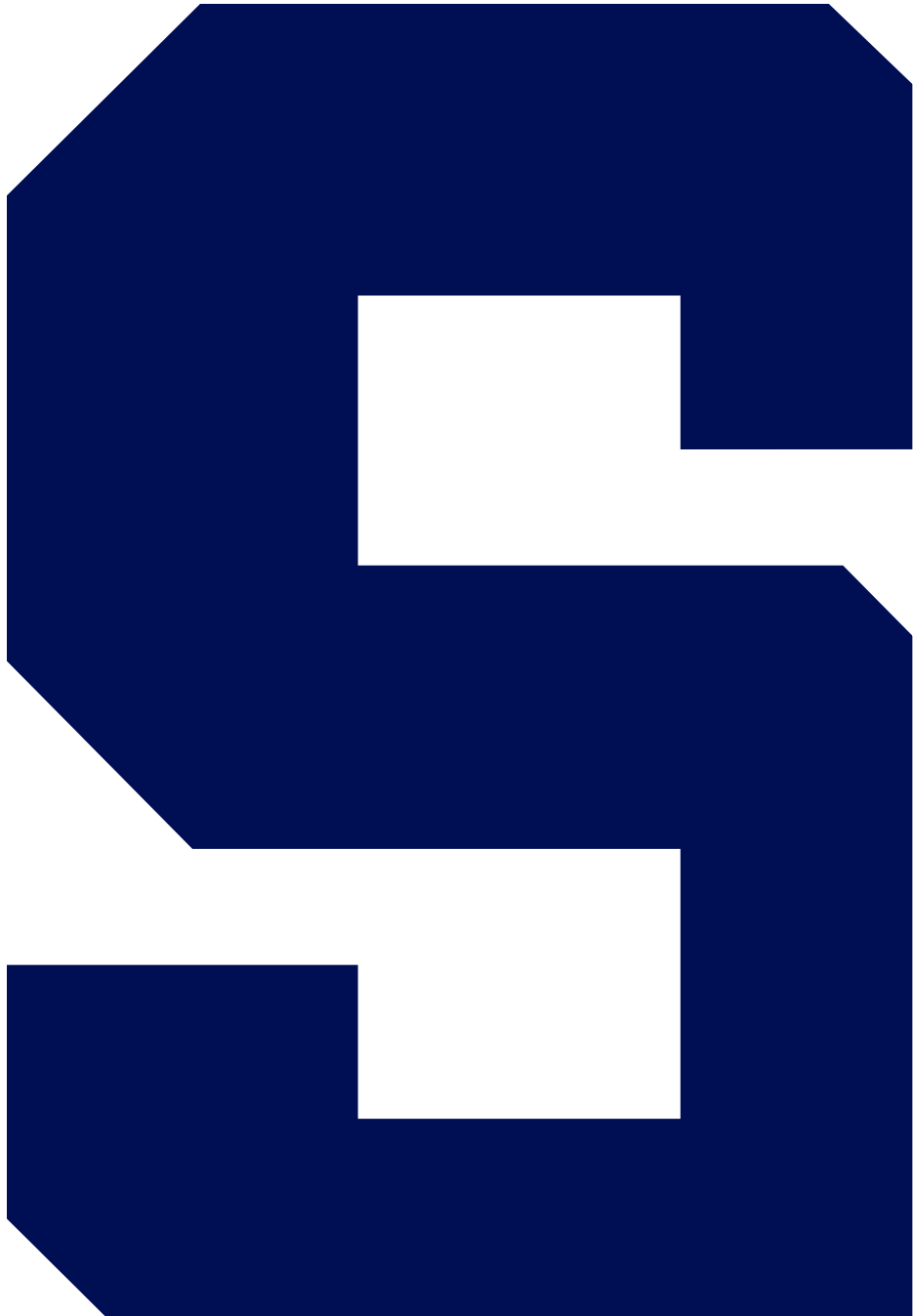
Demo: Index Statistics

- We will use the Azure Data Studio application.
- We will use the demo database.
- Create a table and index.
- Add data—not fragmented is on one page!
- Perform some random updates.
- Check index fragmentation.
- Rebuild and reorganize.

Demo: Index Statistics

The End

Syracuse University
School of Information Studies



Columnstore Indexes

Columnstore Index

- The standard for storing and querying large tables with many columns
- Rather than storing the data in rows, the data are stored in columns
- Appropriate for tables with many columns (wide tables) and those with many similar values, as found in data warehouse fact tables
- Comes in clustered/nonclustered versions
- Data are compressed and cached in memory

Tangent: What Is a Columnstore?

Physical Rowstore

Row 1	US
	Alpha
	3,000
Row 2	US
	Beta
	1,250
Row 3	JP
	Alpha
	700
Row 4	UK
	Alpha
	450

Easier to retrieve a column

Easier to aggregate values or distinct values in a column

Used in analytics

Logical table

Country	Product	Sales
US	Alpha	3,000
US	Beta	1,250
JP	Alpha	700
UK	Alpha	450

Easier to add new rows

Simple to retrieve a row, update a row

Used for CRUD operations

Physical Columnstore

Country	US
	US
	JP
	UK
Product	Alpha
	Beta
	Alpha
	Alpha
Sales	3,000
	1,250
	700
	450
Country	US
	US
	JP
	UK

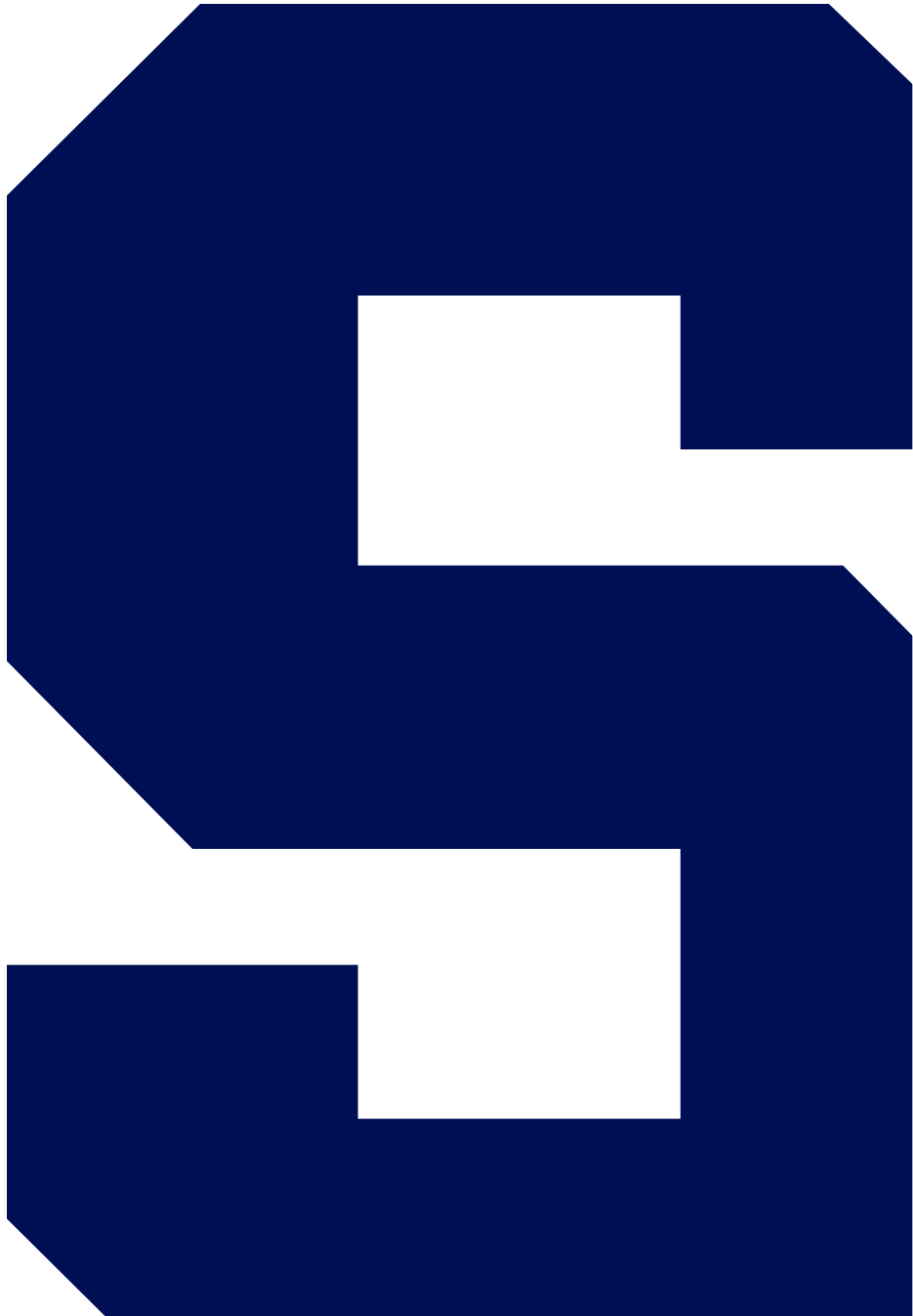
SQL: Create Columnstore

```
CREATE [CLUSTERED|NONCLUSTERED]
    COLUMNSTORE INDEX index_name
    ON table_name (column,[…])
```

- Can be clustered or nonclustered; only one clustered index
- The columns specified are not keys, they are the columns included in the column store index; the keys are the names of the columns!

Columnstore Indexes

The End



Demo

Columnstore Indexes

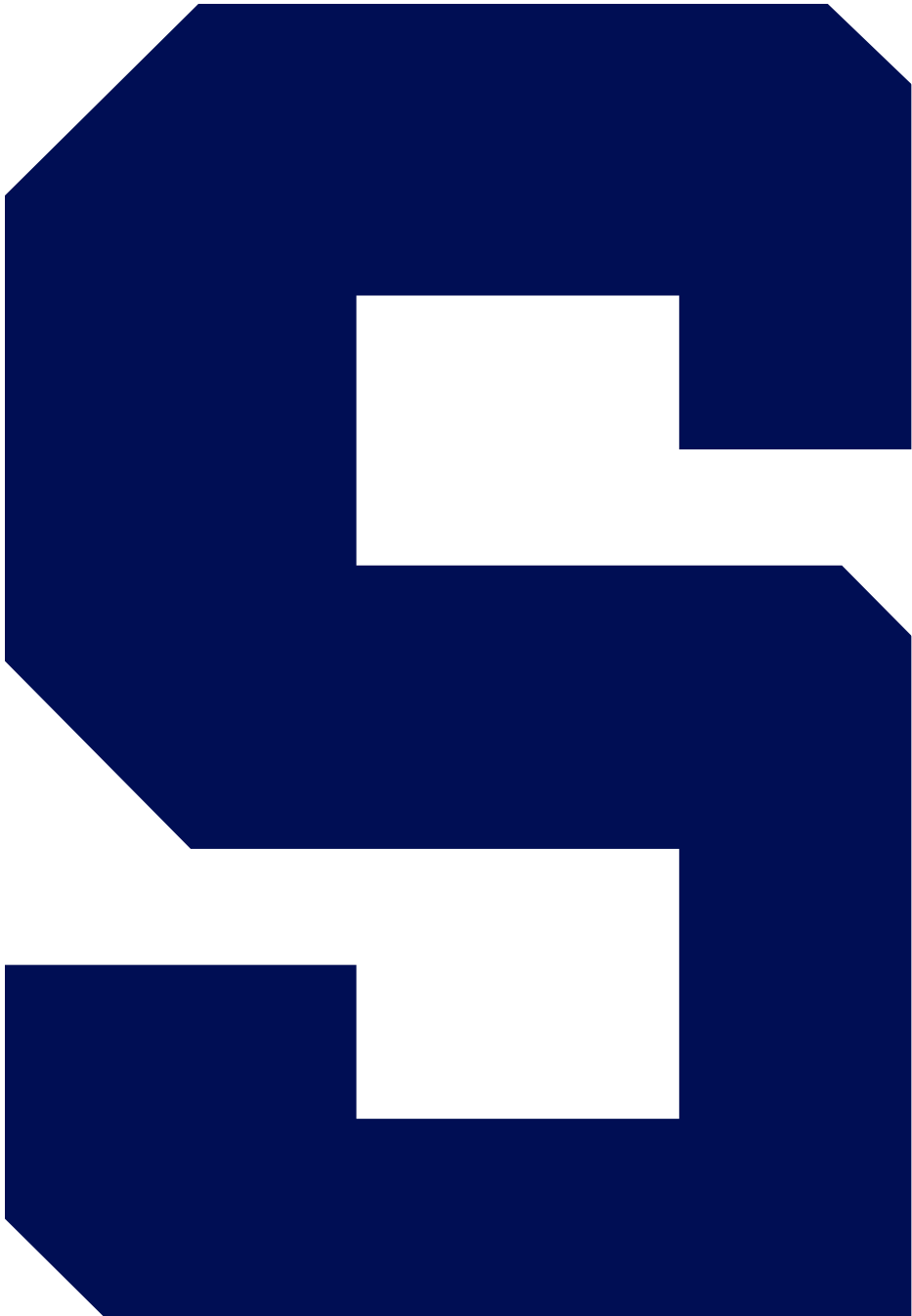
Demo: Columnstore Index

- We will use the Azure Data Studio application
- We will use the fudgemart_v3 database
- Run query on orders table and look at live statistics
- Make a nonclustered index to improve this query
- But it doesn’t work for other queries—we’d need another index
- Columnstore to the rescue!

Demo: Columnstore Indexes

The End

Syracuse University
School of Information Studies



Indexed Views

Indexed Views

- Easy way to improve the performance of a view
- An indexed view is SQL Server's version of a materialized view
- The output of the view is saved into an index table to improve read performance of the view
- When the underlying data are updated, so is the view, just like an index!
- Must bind the schema of the underlying tables—cannot alter the underlying table schema without dropping the view first

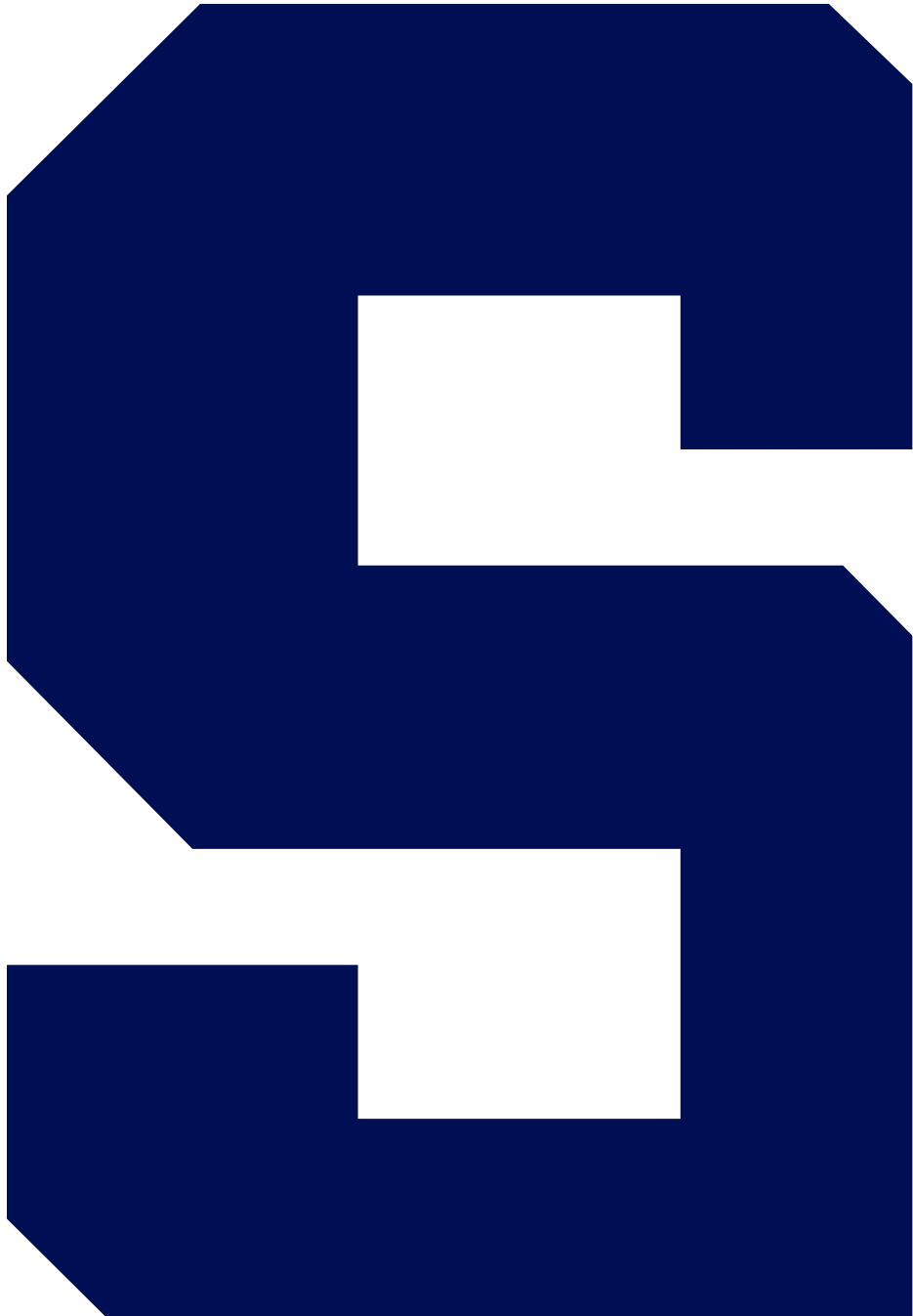
Rules for Indexing a View

- Use WITH SCHEMABINDING in the CREATE VIEW statement
- The view must be deterministic—the same input yields the same output
- All columns must be specified; no wildcard columns
- Tables must be schema-qualified dbo
- Must be a unique clustered index (like setting a PK)

Indexed Views

The End

Syracuse University
School of Information Studies



Demo

Indexed Views

Demo: Indexed Views

- We will use the Azure Data Studio application
- We will use the tinyu database
- Query plan of a view
- Add with schemabinding
- Then index
- Query plan is now like a table!

Demo: Indexed Views

The End

Syracuse University
School of Information Studies

Summary

Physical Performance Tricks and Tips

1

If your server has more than one disk, create one file per disk in the same filegroup to spread IO over the disks.

2

If your server has faster SSD (solid-state disks), create a filegroup on that disk to support high-velocity tables.

3

Do not store the database files on the same disk/partition as the operating system and SQL Server itself. Minimize your IO contention.

4

Put fast-growing time-series data and associative entities in their own filegroup.

Index Rules of Thumb

Use indexes

- The table contains over 100,000 rows.
- The searchable field (indexed column) has a wide range of values.
- The searchable field has a large number of null values.
- The searchable field is queried frequently.
- Queries retrieve less than 2–4% of the table's rows.

Avoid indexes

- There are relatively few rows in the table.
- The column is not used for searching.
- The majority of the queries retrieve more than 2–4% of the table's rows.
- There is high insert or update transaction volatility.

Summary

- The physical model is an implementation of the logical model on a DBMS.
- The SQL Server physical model consists of servers, instances, databases, filegroups, files, and pages.
- Indexes allow us to improve query performance at the expense of write performance.
- Columnstore indexes store table data in columns for faster retrieval.
- Views can be indexed to improve performance.

Syracuse University
School of Information Studies

Summary

The End