Building Effective Indexes

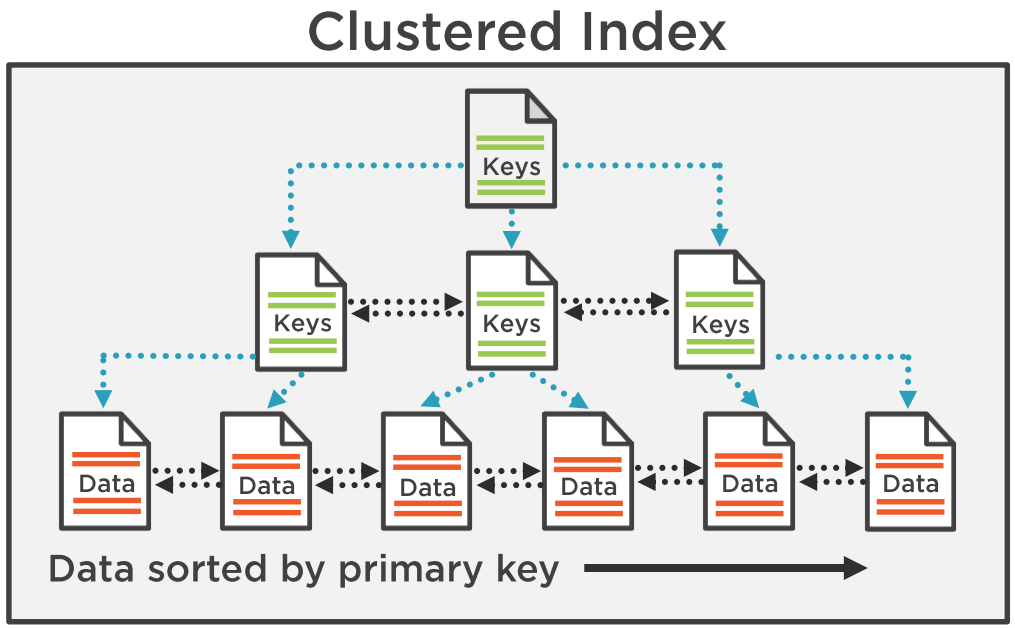
David Berry
@davidcberry13 buildingbettersoftware.blogspot.com

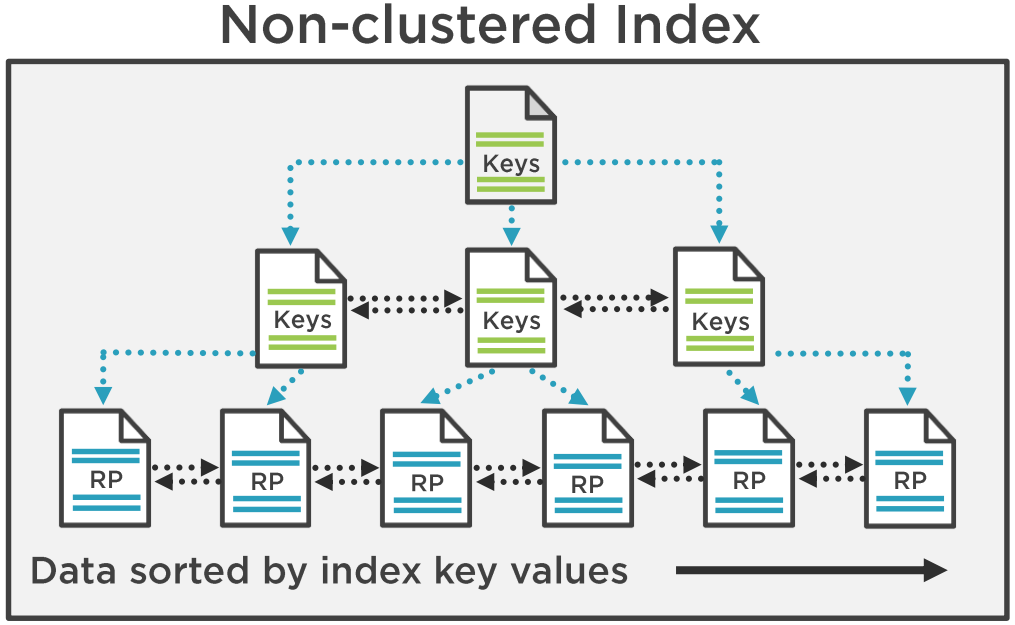
Module Outline

- Index terminology refresher
- What columns should I index?
- Why column order matters in an index?
- Index selectivity
- Include columns and covering indexes
- Functions in the WHERE clause and indexes
- Over-indexing
- SQL Server's index recommendations

Index Terminology Refresher

Clustered vs. Non-clustered Indexes





Stores the data for the table

Typically organized by the primary key

Only one allowed per table

Structure used for all other indexes

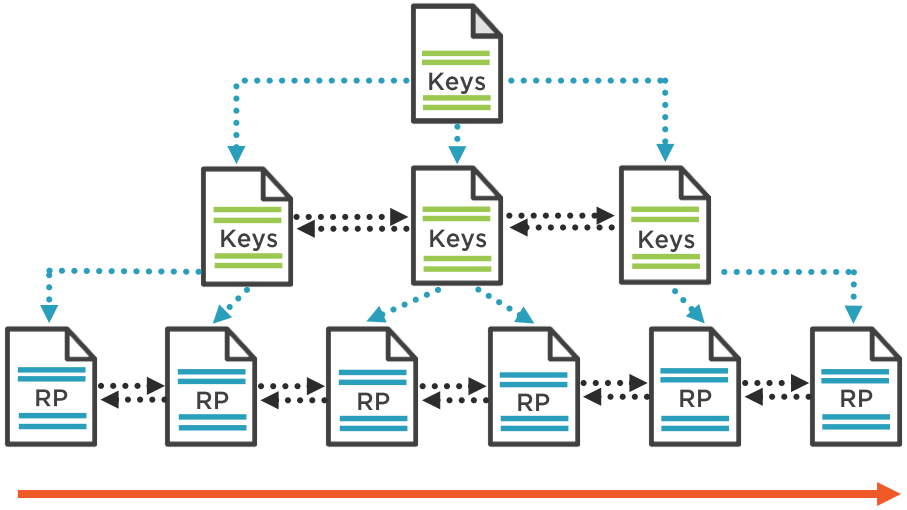
Data organized by index key

Contains pointers to matching rows

Multiple allowed per table

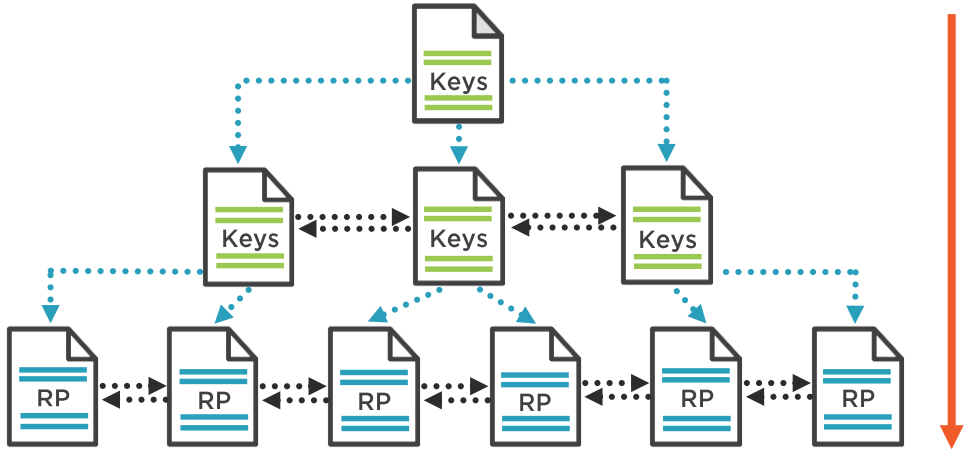
Scan vs. Seek Operations

Index Scan Operation



SQL Server reads all data in the index

Index Seek Operation



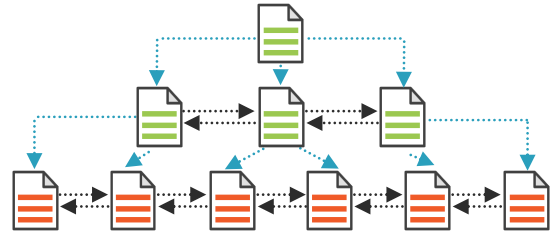
SQL Server traverses the index using the tree structure

Indexing Strategy

By correctly indexing columns up front, you can save yourself from most performance problems later

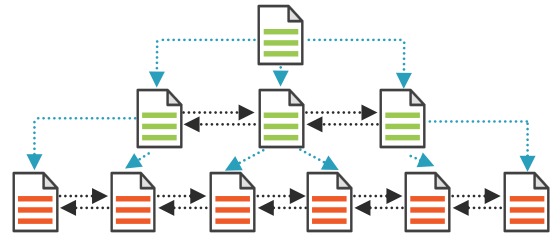
Add Indexes for WHERE Clause Criteria

```
SELECT *
  FROM Students
  WHERE LastName = 'Howard'
    AND FirstName = 'Emily'
```



Index on LastName, FirstName

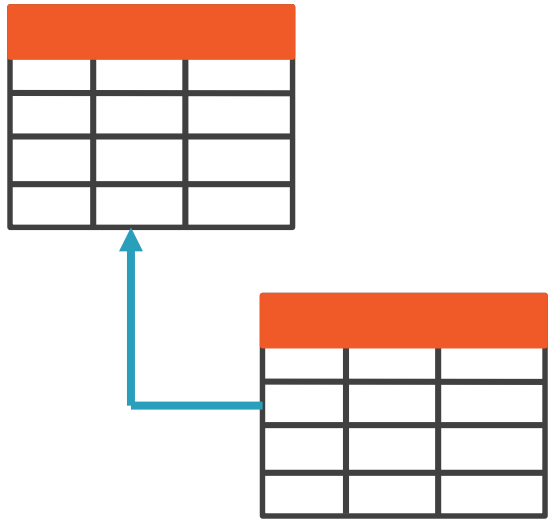
```
SELECT *
  FROM Students
  WHERE Email =
    'Ehoward@someuniversity.edu'
```



Index on Email

Primary key columns are indexed by default in SQL Server

Index Foreign Key Columns



Helps speed up join operations

Applications often query tables by foreign key values

Applying Index Column Order Lessons

How Data is Queried

```
SELECT * FROM Students
  WHERE LastName = ?????
```

```
SELECT * FROM Students
  WHERE LastName = ????? AND FirstName = ?????
```

```
SELECT * FROM Students
  WHERE LastName = ????? AND FirstName = ?????
    AND State = ?????
```

Index IX_Students_NameSearch

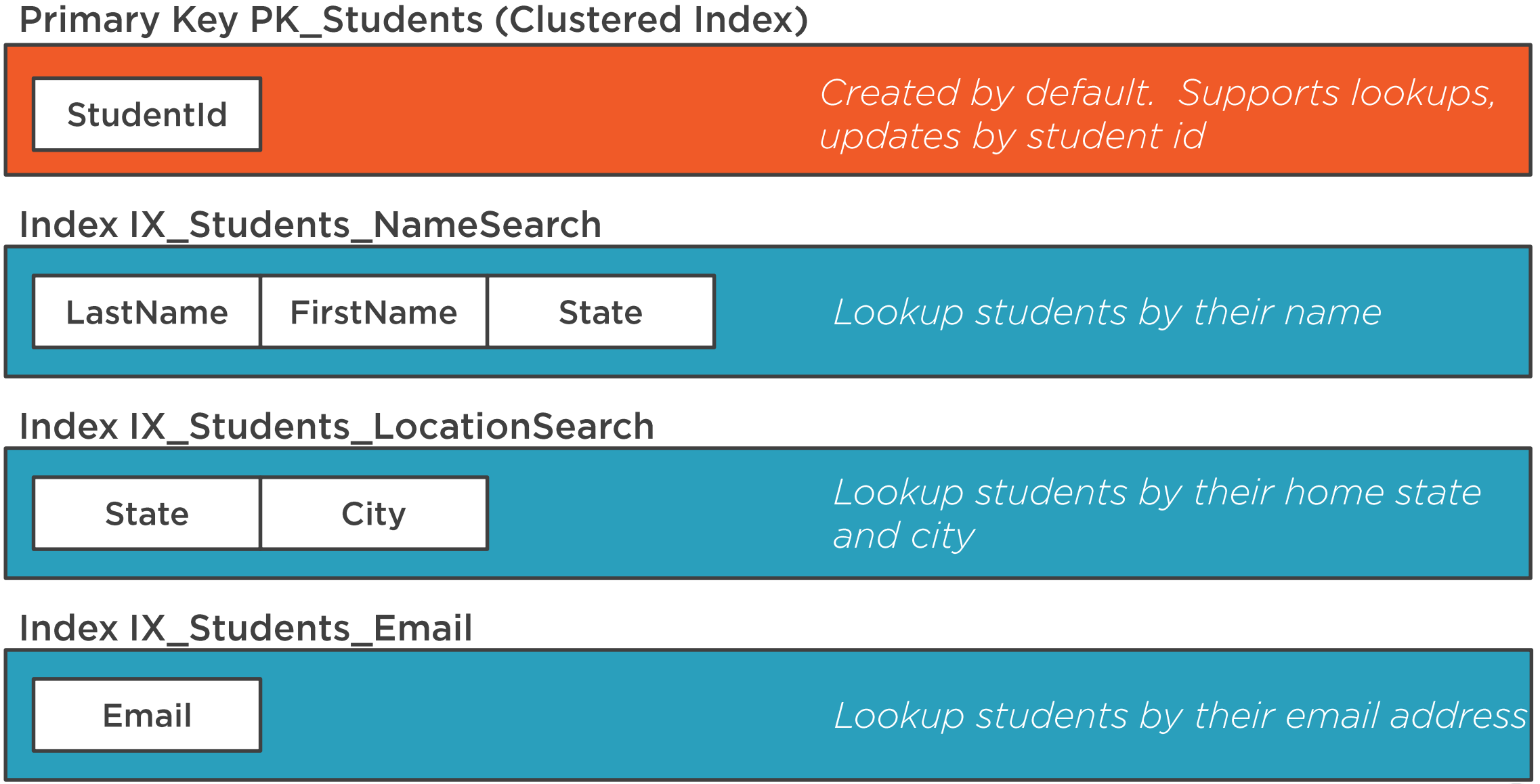
LastName	FirstName	State

```
SELECT * FROM Students
  WHERE State = ????? AND City = ?????
```

Index IX_Students_LocationSearch

State	City

Student Table Indexes

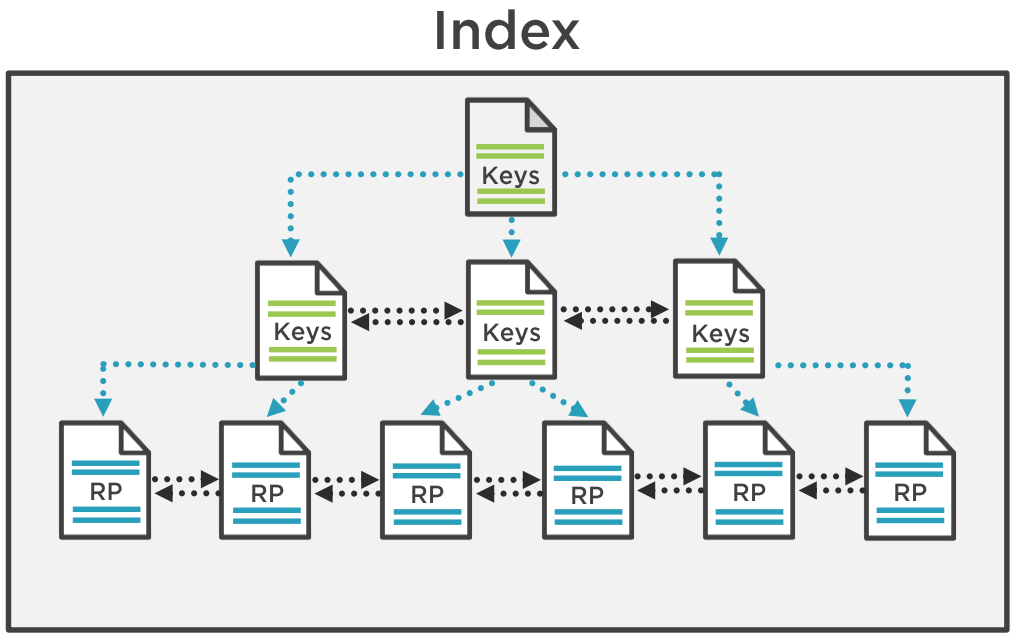


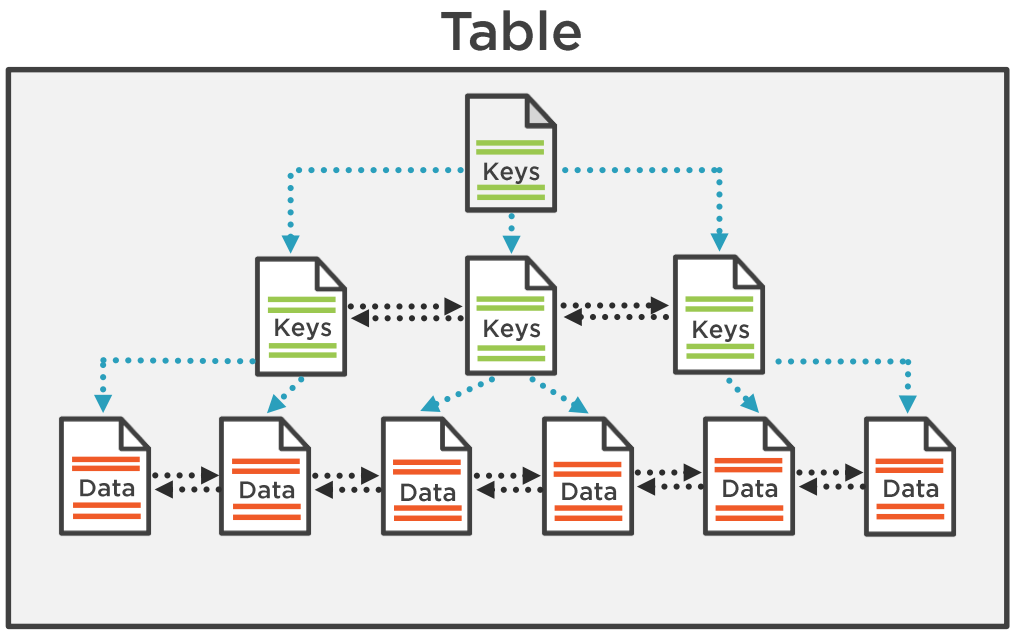
Index Selectivity

A measure of how many or how few rows correspond to each index key value

For SQL Server to use an index, it must be selective

Index Operations





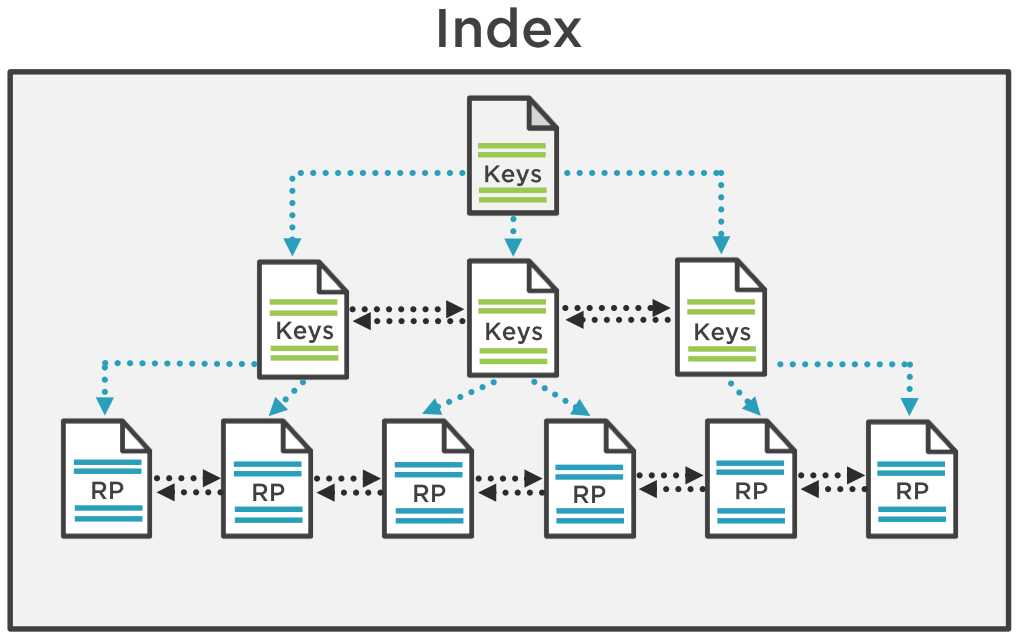
SQL Server looks up matching index keys, gets row identifiers

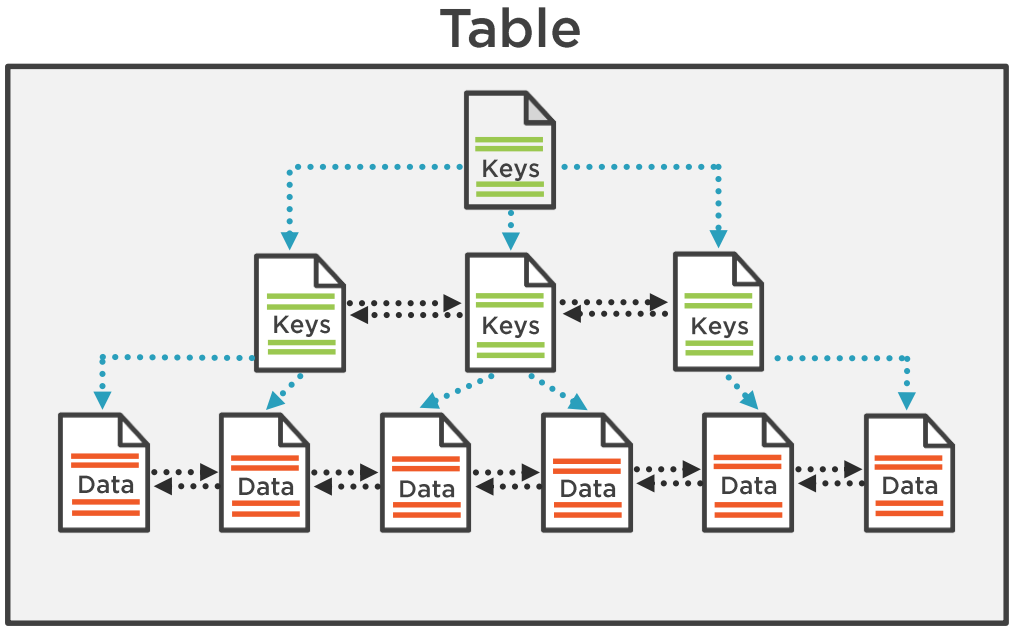
SQL Server looks up corresponding rows by their row identifier

Effects of Index Selectivity

	Index Operation	Table Lookup Operation	Overall Cost
Index is Selective	Few rows for each matching key	Few rows to lookup	Low. Very few pages to read
Index is not Selective	Many rows for each matching key	Many rows to look up	High. Many pages to read

Index Lookup Operations





SQL Server finds matching keys

Index is Selective

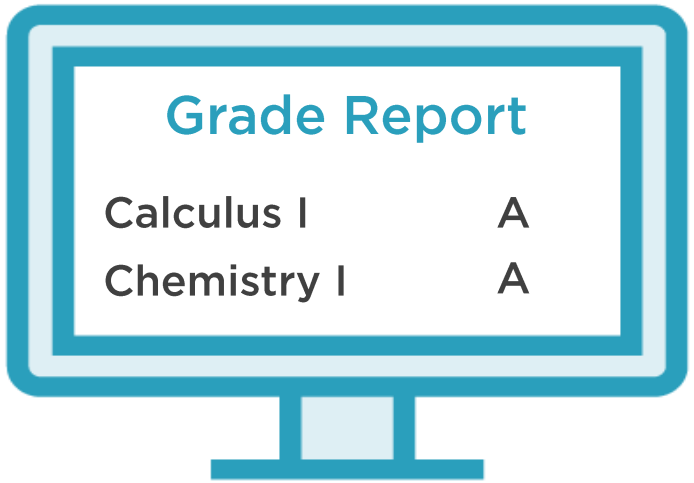
Index is not Selective

Selective indexes help SQL Server locate the exact data needed with minimal effort

Statement Performance Comparison

Metric	Chosen by SQL Optimizer	Index Usage Forced with Hint
Data Access Method	Table scan	Index seek
Statement Cost	2.903	5.826
Logical Reads	3747	7189

Queries Often Follow Foreign Keys



Student ID = 12345

```
SELECT c.DepartmentCode,
    c.CourseNumber, c.CourseTitle,
    ce.Grade,
  FROM CourseEnrollments ce
    ...
  WHERE ce.StudentId = 12345
```

Covering Index

SQL Server is able to find all the data for the query in the index itself

This avoids needing a key lookup operation to find the data row in the table

Consider a covering index when you only need to add one or two columns as include columns to give a performance boost to a key query

Question:

If indexes are so great, then why not create an index on every column?

Answer:

Indexes require maintenance, so you only want to create indexes that are actually used by your SQL statements

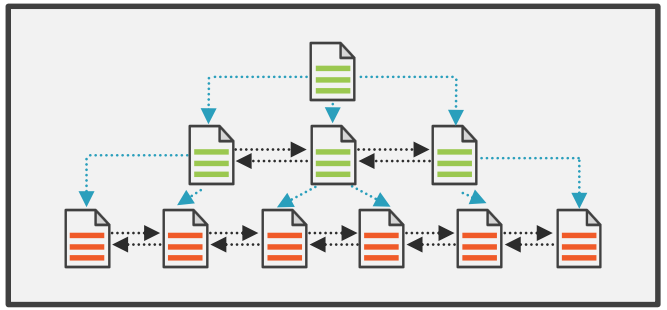
Table Data

Id	FirstName	LastName	Email
101	Mark	Gilbert	mgilbert@...
102	Paul	Wilson	pwilson@...
103	Erica	Griffin	egriffin@...
105	Peter	Wright	pwright@...

DML Statement
(INSERT, UPDATE or DELETE)

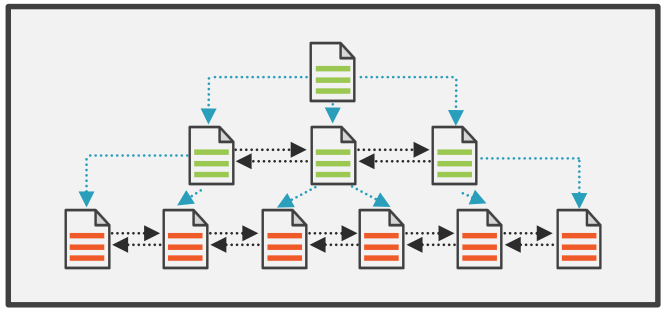
Index update

Index



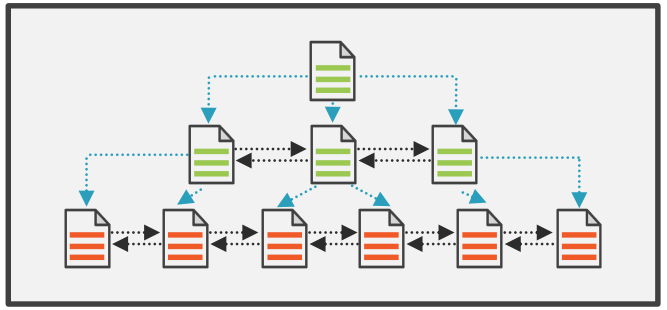
Index update

Index



Index update

Index



Regard Indexes as an Investment

Indexes that are frequently used repay this investment many times

Indexes that are seldom or never used add cost without adding value

SQL Server Dynamic Management Views provide access to a rich set of performance data, including index usage data

Finding Unused Indexes Using DMVs

```
SELECT
    OBJECT_NAME(s.object_id) AS TableName,
    i.name As IndexName,
    i.type_desc as IndexType,
    user_seeks + user_scans + user_lookups  As TotalUsage,
    user_seeks,
    user_scans,
    user_lookups,
    user_updates
FROM sys.dm_db_index_usage_stats s
RIGHT OUTER JOIN sys.indexes i
    ON s.[object_id] = i.[object_id]
    AND s.index_id = i.index_id
WHERE s.database_id = DB_ID()
    AND i.name IS NOT NULL
    AND OBJECTPROPERTY(s.[object_id], 'IsMsShipped') = 0
ORDER BY s.object_id, s.index_id
```

SQL Server Index Recommendations

Analyze the recommendation first before creating

SQL Server can be overly aggressive

SQL Server's recommendations are based on looking at one individual SQL statement

You want to view these recommendations in the larger context of all of your statements for a given table

Module Review

Index Columns Used in WHERE Clauses

```
SELECT *
    FROM Students
    WHERE LastName = ????
        AND FirstName = ????
```

```
SELECT *
    FROM Students
    WHERE Email = ????
```

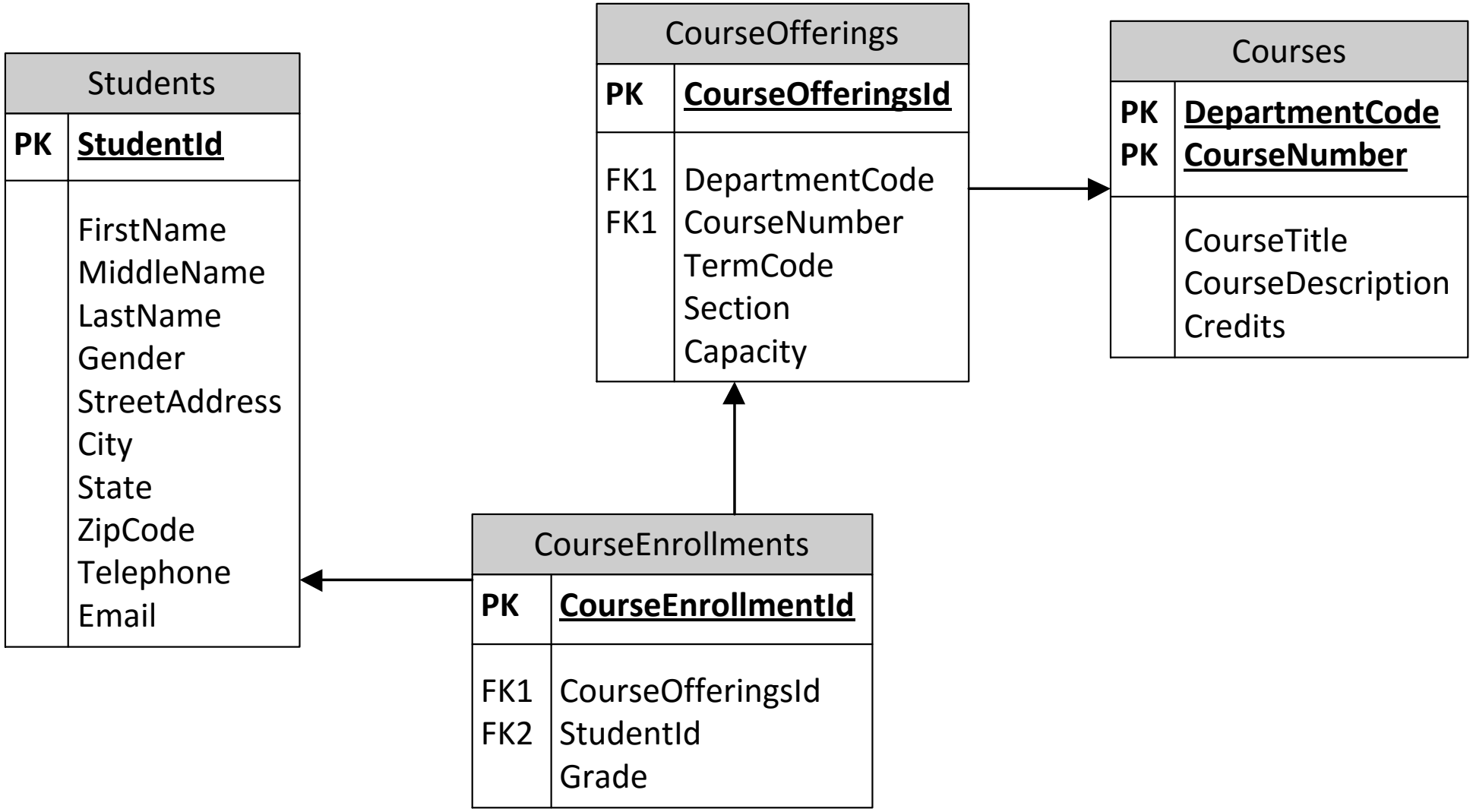
```
SELECT *
    FROM Students
    WHERE State = ????
        AND City = ????
```

```
SELECT *
    FROM Applicants
    WHERE Email = ????
```

```
SELECT *
    FROM Applicants
    WHERE LastName = ????
        AND FirstName = ????
```

```
SELECT *
    FROM CourseOfferings
    WHERE TermCode = ????
        AND DepartmentCode = ????
        AND CourseNumber = ????
```

Index Foreign Key Columns

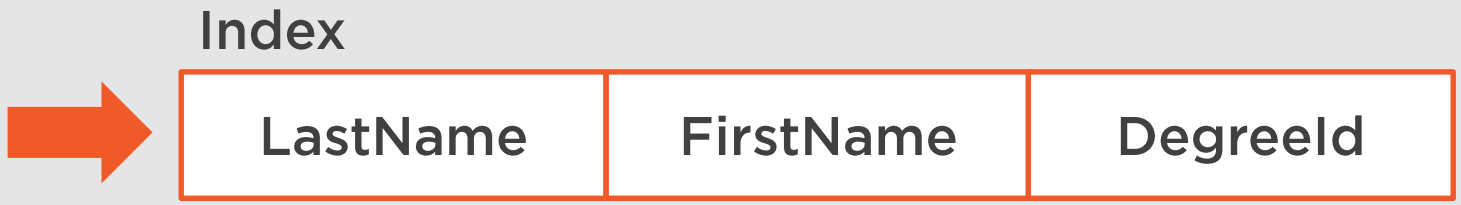


Column Order in Indexes Matters

```
SELECT *
    FROM Students
    WHERE LastName = ????
        AND FirstName = ????
```

```
SELECT *
    FROM Students
    WHERE LastName = ????
```

```
SELECT *
    FROM Students
    WHERE LastName = ????
        AND DegreeId = ????
```



Index Selectivity

Index

State

$$\frac{\text{120,000 rows}}{\text{50 unique values}} = \text{2400 rows/index key}$$

Poor Selectivity

Index

State	City

$$\frac{\text{120,000 rows}}{\text{13,000 unique values}} = \text{9.2 rows/index key}$$

Good Selectivity

Index

LastName	FirstName

$$\frac{\text{120,000 rows}}{\text{107,000 unique values}} = \text{1.1 rows/index key}$$

Good Selectivity

LIKE Clauses

```
-- This statement will not use an index
SELECT *
    FROM Students
    WHERE LastName = '%Harris%' AND FirstName = '%Christy%'

-- This statement will use an index
SELECT *
    FROM Students
    WHERE LastName = 'Harris%' AND FirstName = 'Christy%'

-- But beware of statements like this where you don't
-- provide much information.  It probably will not use an index
SELECT *
    FROM Students
    WHERE LastName = 'Ha%' AND FirstName = 'Ch%'
```

Functions in the WHERE Clause

```
-- Statements like this will not use a normal index
SELECT *
    FROM Students
    WHERE SOUNDEX(LastName) = SOUNDEX('SMITH')
        AND SOUNDEX(FirstName) = SOUNDEX('CHRIS');


-- But we can create a computed column and an index over the
computed column if we need this functionality
ALTER TABLE Students ADD LastNameSoundex AS Soundex(LastName);
ALTER TABLE Students ADD FirstNameSoundex AS Soundex(FirstName);

CREATE INDEX IX_Students_PhoneticName
    ON Students (LastNameSoundex, FirstNameSoundex);
```

Up Next

Finding Performance Bottlenecks in SQL Server