



ColdFusion Summit 2016



Building Better SQL Server Databases

Who is this guy?



Eric Cobb

- Started in IT in 1999 as a "webmaster"
- Developer for 14 years
- Microsoft Certified Solutions Expert (MCSE)
 - Data Platform
 - Data Management and Analytics
- Now a full time DBA

Blog: <http://www.sqlnuggets.com>

Blog Twitter: @sqlnugg

Personal Twitter: @cfgears



What are we going to learn?

- How SQL Server stores and retrieves data
- How your database design can impact resources and performance
- Design tips to incorporate into your database development
- Common T-SQL mistakes and things to avoid
- Primarily focusing on OLTP databases



A PEEK UNDER THE HOOD OF SQL SERVER

A BRIEF OVERVIEW OF HOW SQL SERVER STORES AND RETRIEVES DATA

A Peek Under The Hood

Tables & Data Pages

	EmployeeId	LastName	Country	DepartmentId
1	123	Penn	Australia	31
2	124	Fitch	Australia	33
3	145	Jackson	Australia	33
4	201	Shields	United States	34
5	305	Pierre	Germany	34
6	306	Velasquez	Germany	NULL
7	310	Lawal	United States	31
8	311	Bell	China	35
9	312	George	Japan	36

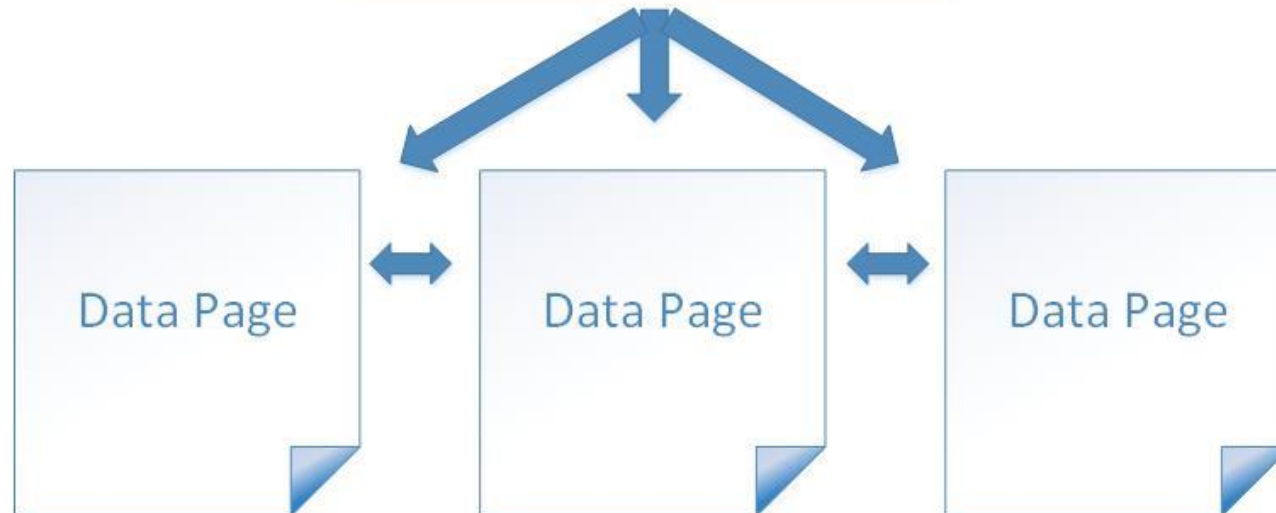


Data Page

A Peek Under The Hood

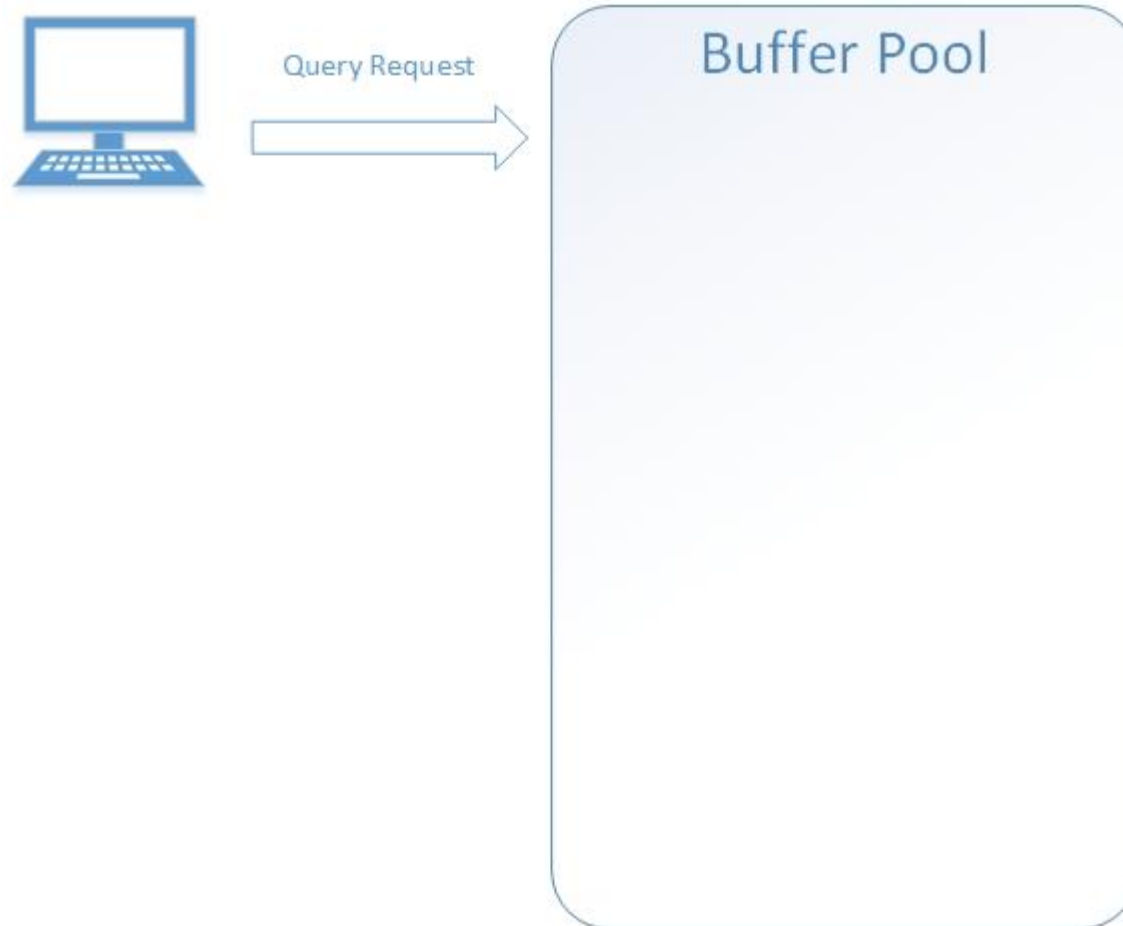
Tables & Data Pages

dbo.Employee				
	EmployeeId	LastName	Country	DepartmentId
1	123	Perrin	Australia	31
2	124	Fitch	Australia	33
3	145	Jackson	Australia	33
4	201	Shields	United States	34
5	305	Pierre	Germany	34
6	306	Velasquez	Germany	NULL
7	310	Lawai	United States	31
8	311	Bell	China	35
9	312	George	Japan	36



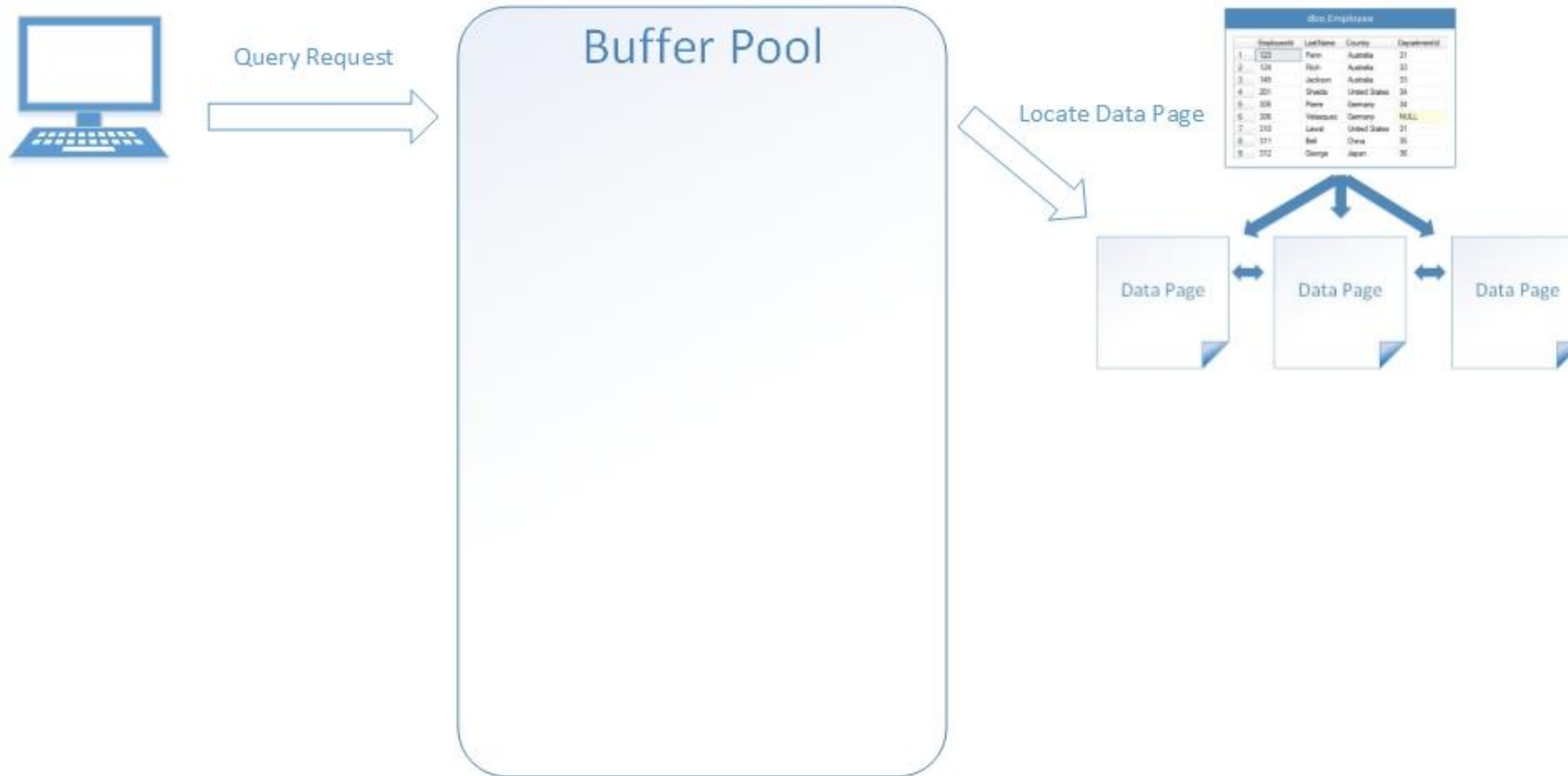
A Peek Under The Hood

Process of Requesting Data



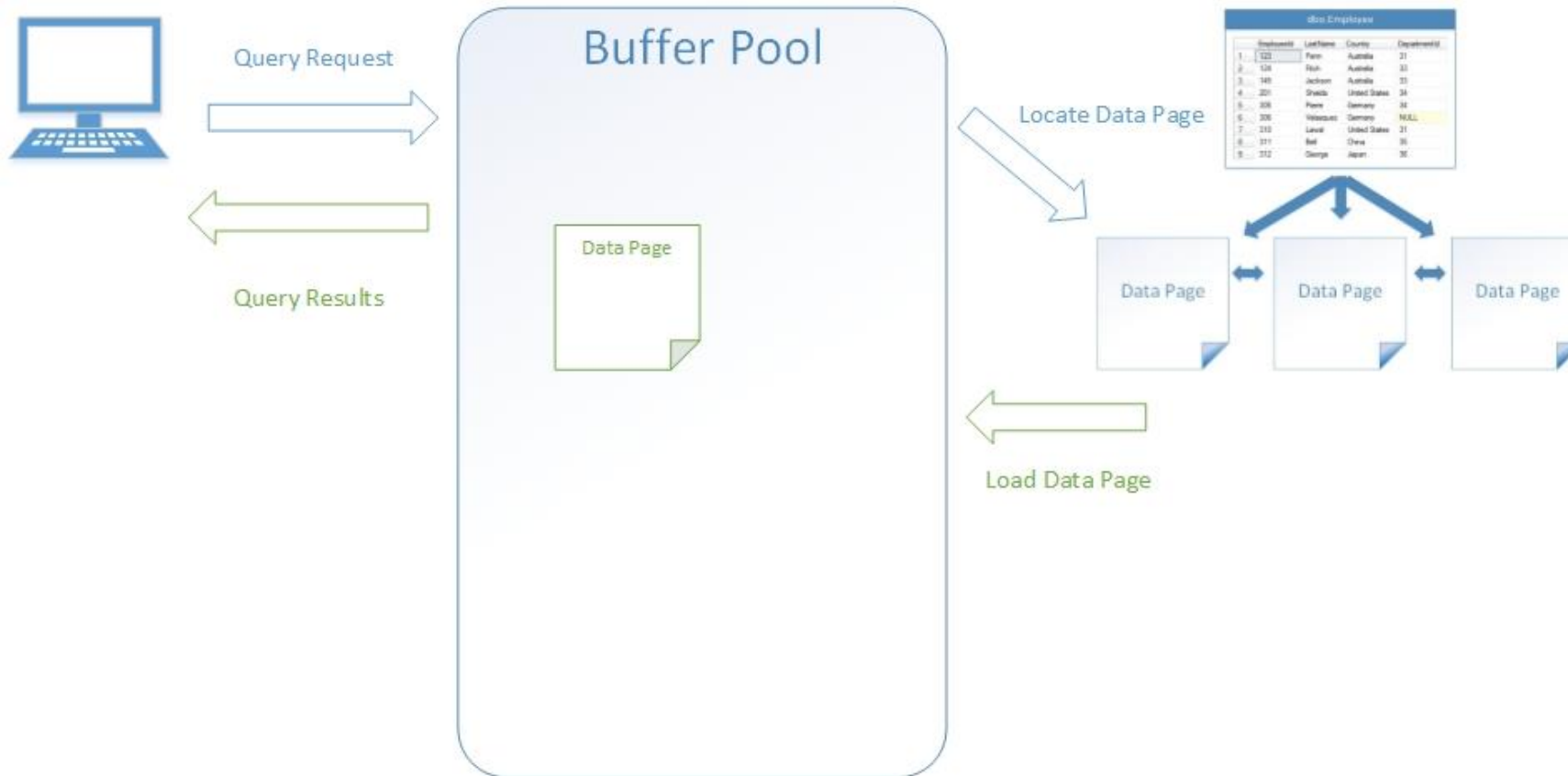
A Peek Under The Hood

Process of Requesting Data



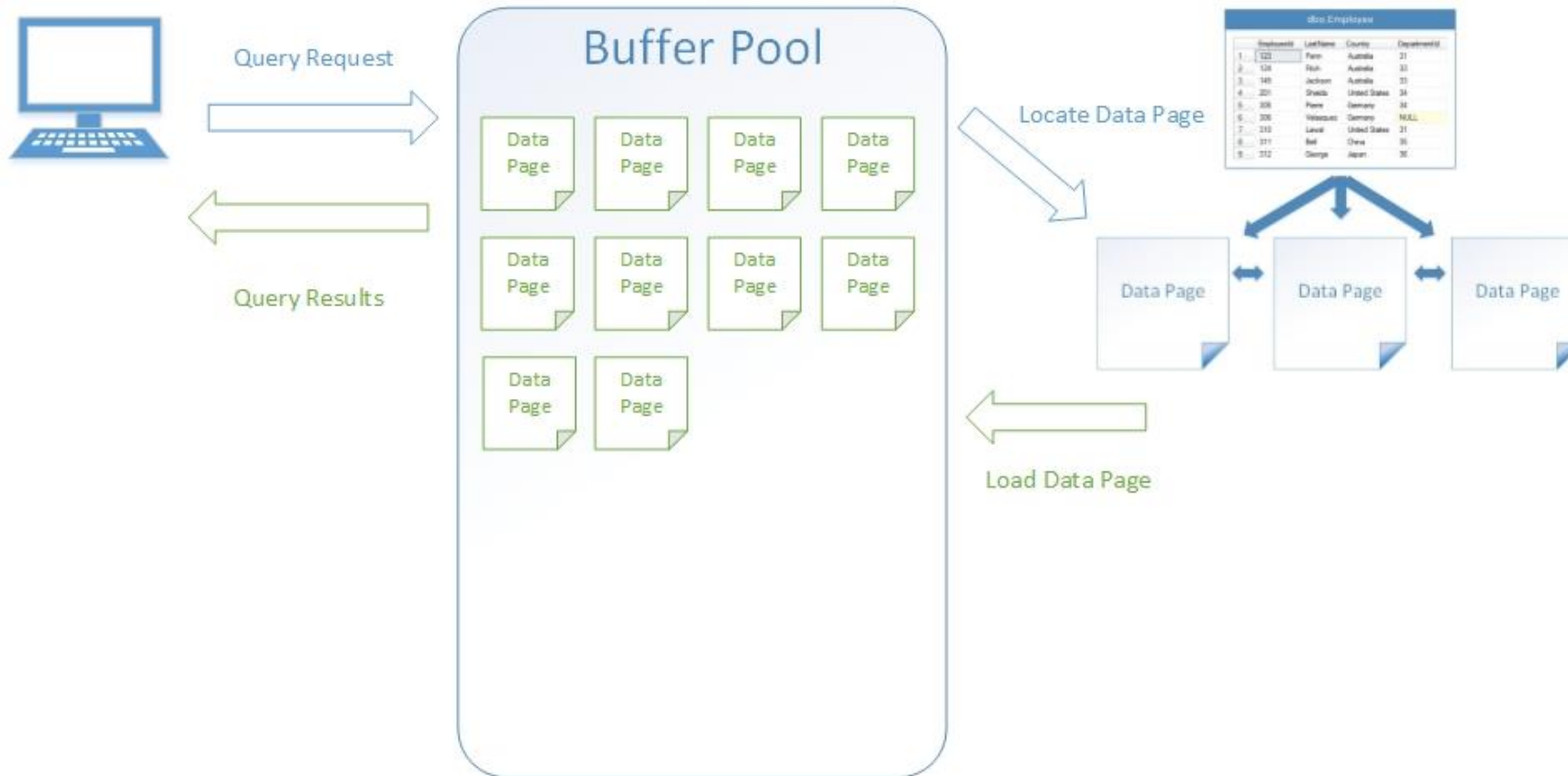
A Peek Under The Hood

Process of Requesting Data



A Peek Under The Hood

Process of Requesting Data

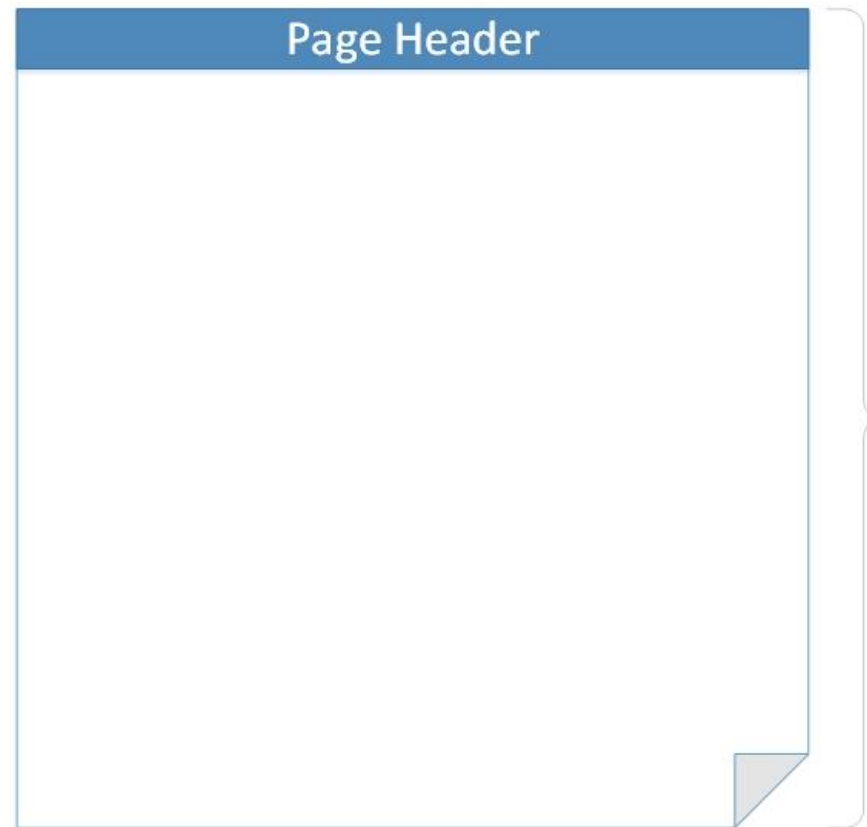


A Peek Under The Hood

Storing Data in Pages



Data Page



8 KB

A Peek Under The Hood

Storing Data in Pages



Data Page



A Peek Under The Hood

Storing Data in Pages



How is the data stored in a Page?

Unordered (Heap)

- Query optimizer reads all the rows in the table (table scan), to find the rows that meet the criteria of a query
- A table scan generates many disk I/O operations and can be resource intensive
- Heaps should generally be avoided, although can be useful when inserting large amounts of data in ETL/Bulk processes

Ordered (Clustered Index)

- Tells SQL Server how to physically sort the records on disk
- The most important index you can apply to a table
- Data pages are ordered, for faster data retrieval
- Query optimizer searches the index key columns and finds the location of the rows needed by the query
 - Searching the index is much faster than scanning the entire table
- There is only ever 1 clustered index on a table

A Peek Under The Hood

Storing Data in Pages



How do I create Clustered Indexes?

Primary Key = Clustered Index (usually)

- SQL Server automatically creates a clustered index on your Primary Key column if a clustered index does not already exist on the table
- If you do not want the Primary Key to be your Clustered Index, you can create your Clustered Index on a different column

Clustered Index (Primary Key) Tips:

- Use a naturally occurring incremental value
- Keep as small and narrow as possible (single columns are preferred)
- Avoid using character data types for a Clustered Index

A Peek Under The Hood

Storing Data in Pages



Page Header						
LastName					
Adams						
Allan						
Barnes						
Davis						
Franklin						

A Peek Under The Hood

Page Splits

Page Header				
LastName	****			
Adams				
Allan				
Barnes				
Cobb				



Page Header				
LastName	****			
Davis				
Franklin				

A Peek Under The Hood

Page Splits

Page Header			
Adams			
Allan			
Barnes			
Cobb			



Page Header			
LastName		
Davis			
Franklin			

A Peek Under The Hood

Page Splits



The diagram illustrates a page split in a table. Two identical table structures are shown side-by-side, connected by a blue double-headed arrow. Each table has a blue header row labeled 'Page Header' and a grey row labeled 'LastName' with a column width indicator '....'. The left table contains the following data rows: Adams, Allan, Barnes, Cobb, Cuthbert, and Danson. The right table contains the following data rows: Davis and Franklin. Both tables have a white background and a blue border.

Page Header	
LastName
Adams	
Allan	
Barnes	
Cobb	
Cuthbert	
Danson	

Page Header	
LastName
Davis	
Franklin	

A Peek Under The Hood

Page Splits



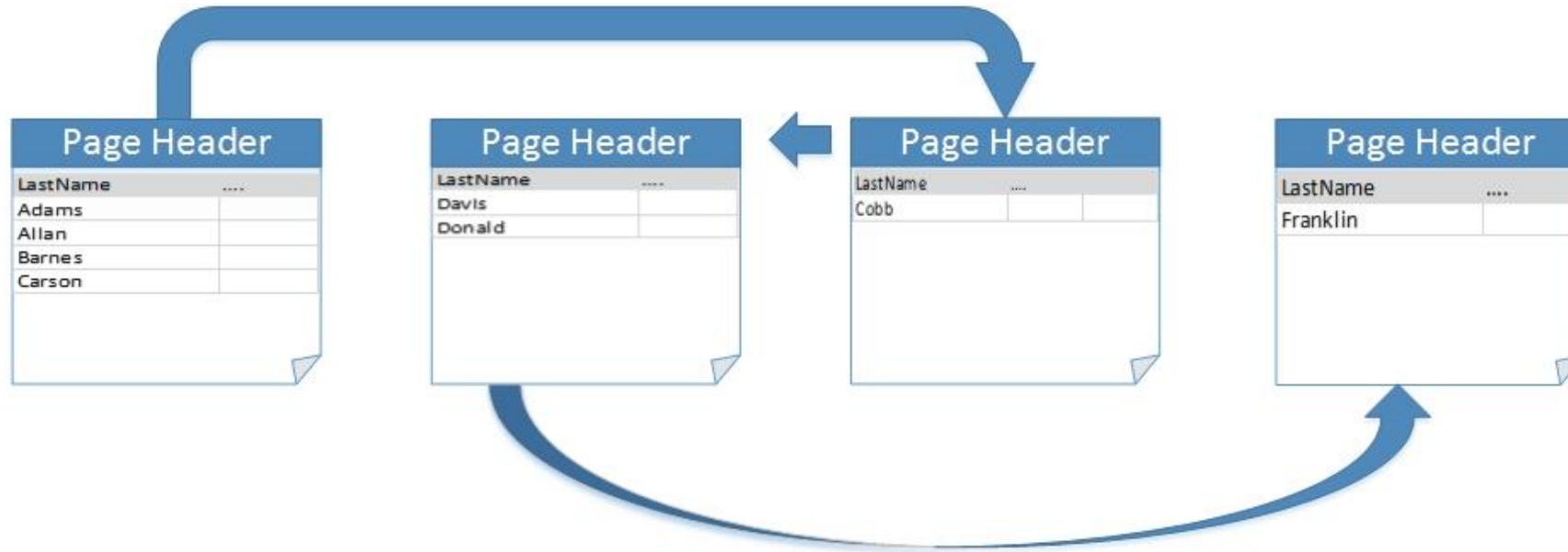
A Peek Under The Hood

Page Splits



A Peek Under The Hood

Page Splits



A Peek Under The Hood

Page Splits



How can we avoid Page Splits?

You can't avoid them, but you can minimize them with good table designs

- Choose a good Clustered Index (Primary Key) for your table
 - Should be unique, narrow, static, and incremental
 - Good Clustered Index examples:
 - A numeric identity column (smallint, int, bigint)
 - A composite key of date and identity – in that order (date, identity)
 - A pseudo sequential GUID (using the NEWSEQUENTIALID() function in SQL Server)
 - Not recommended, but the best you can do if you absolutely have to use a GUID
 - Clustered Indexes to avoid:
 - Unique Identifier (GUID) generate from an application or with SQL Server's NEWID() function
 - Character columns (CHAR, VARCHAR, NVARCHAR, etc...)
 - Combination of multiple character columns (LastName, FirstName, MiddleInitial)
 - Columns that undergo frequent changes

A Peek Under The Hood

Overflow Pages



Data Page



A Peek Under The Hood

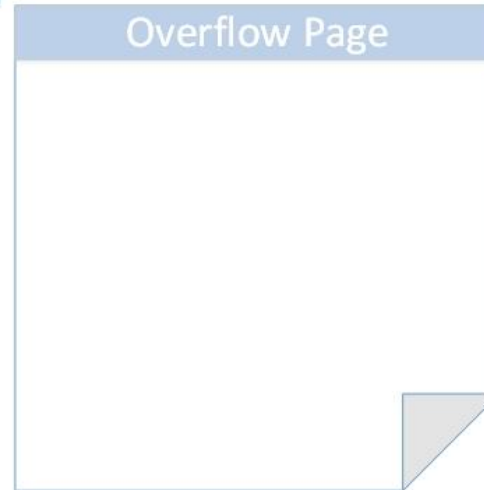
Overflow Pages

Data Page

Page Header
Data Row 1
Data Row 2
Data Row 3
Data Row 4
Data Row 5
Data Row 6

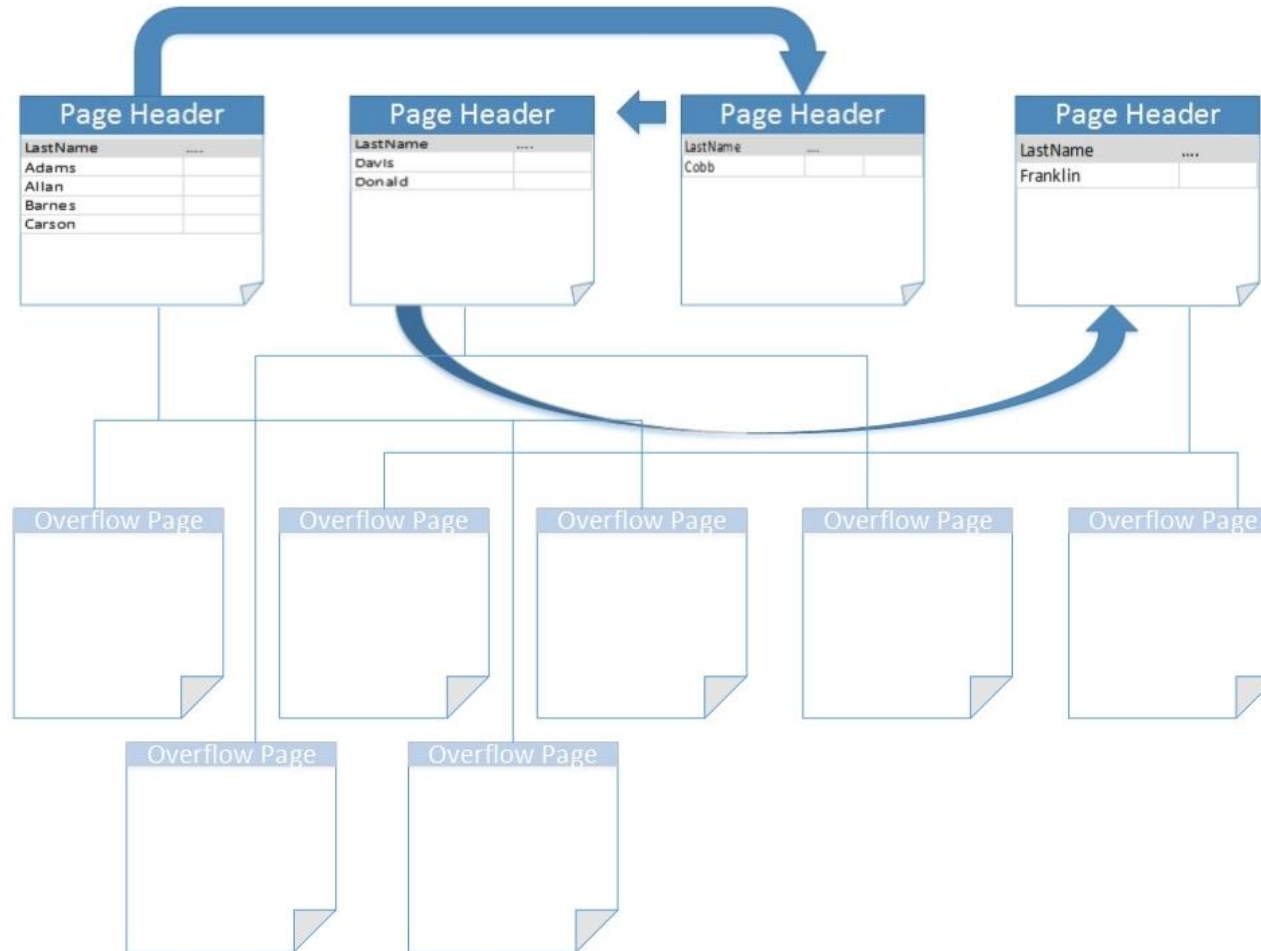


Overflow Page



A Peek Under The Hood

Overflow Pages



The background is a vibrant green with a low-poly, crystalline texture. A large, 3D 'cf' logo is positioned in the upper right, with the 'c' in a rainbow gradient and the 'f' in green. The logo is set against a white, angular shape that resembles a stylized 'A' or a cluster of triangles. The overall composition is dynamic and modern.

BUILDING BETTER TABLES

DESIGNING TABLES WITH EFFICIENCY IN MIND

Building Better Tables

Using The Right Data Types



Data Types Are Important!

Choose your table column data types wisely

- They can affect the performance of your database as it grows

Know your data, use the appropriate data type for the data you are storing

- The more accurate your data type is, the more efficiently SQL Server can handle your data.

Use the smallest data type possible (within reason)

- The smaller the column, the less data you have to store and retrieve, which leads to faster reads and writes
- The longest city name in the U.S. is *Rancho Santa Margarita, California*; it's 22 chars, don't use VARCHAR(MAX)
- The true name of Bangkok, Thailand is: *Krungthepmahanakhon Amonrattanakosin Mahintharayutthaya Mahadilokphop Noppharatratchathaniburirom Udomratchaniwetmahasathan Amonphimanawatansathit Sakkathattiyawitsanukamprasit*. (176 chars)

Building Better Tables

Using The Right Data Types



CHAR vs VARCHAR

- CHAR(n): Fixed-length string data, and the storage size is n bytes.
- VARCHAR(n): Variable-length string data, the storage size is the actual length of the data entered + 2 bytes.
- If you know the length of the string will always be the same, use CHAR to avoid the additional 2 bytes added to every VARCHAR record

NCHAR vs NVARCHAR

- If you have databases that support multiple languages, consider using the Unicode NCHAR or NVARCHAR data types to minimize character conversion issues
- Carefully evaluate whether you really need NCHAR or NVARCHAR
- NCHAR(n): Fixed-length Unicode string data, and the storage size is two times n bytes
- NVARCHAR(n): Variable-length Unicode string data, and the storage size, in bytes, is two times the actual length of data entered + 2 bytes

Building Better Tables

Using The Right Data Types



DECLARE

```
@var1 CHAR(10) = 'abc',  
@var2 NCHAR(10) = 'abc',  
@var3 VARCHAR(10) = 'abc',  
@var4 NVARCHAR(10) = 'abc'
```

SELECT

```
DATALENGTH(@var1) AS [char],  
DATALENGTH(@var2) AS [nchar],  
DATALENGTH(@var3) AS [varchar],  
DATALENGTH(@var4) AS [nvarchar]
```

	char	nchar	varchar	nvarchar
1	10	20	3	6

Building Better Tables

Using The Right Data Types



Numeric Data Types

Data Type	Range	Storage
BIGINT	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 (Quintillion)	8 Bytes
INT	-2,147,483,648 to 2,147,483,647 (Billion)	4 Bytes
SMALLINT	-32,768 to 32,767	2 Bytes
TINYINT	0 to 255	1 Byte

Choose the appropriate Data Type for the range of numbers you will be storing

Building Better Tables

Using The Right Data Types



Date and Time Data Types

Data Type	Range	Storage
TIME	00:00:00.0000000 through 23:59:59.9999999	3 - 5 Bytes
DATE	0001-01-01 through 9999-12-31	3 Bytes
SMALLDATETIME	1900-01-01 through 2079-06-06	4 Bytes
DATETIME	1753-01-01 through 9999-12-31	8 Bytes

Choose the appropriate Data Type for the range of dates you will be storing

Building Better Tables

Using The Right Data Types



Why does this matter?

Performance

- Smaller data sets = faster queries
- Optimized data pages = optimized resource usage (Remember the Buffer Pool?)

Scalability

- Helps you build better, more scalable applications
 - Don't think in terms of 1 row of data, think about millions
- Scalable applications do not happen by accident

Time spent on proper database design is well worth it

- Minor changes can have a major impact
- It can take more effort to rebuild an existing application than it does to originally design one correctly

Building Better Tables

Using The Right Data Types

Why does this matter?

- For a large table, even just saving 8 bytes can make a difference...
- You must think about scale:

Bytes Saved	Rows	MB Saved	
8	10,000,000	76	7.6 GB
8	100,000,000	763	
8	1,000,000,000	7629	
16	10,000,000	153	15.3 GB
16	100,000,000	1526	
16	1,000,000,000	15259	
32	10,000,000	305	30.5 GB
32	100,000,000	3052	
32	1,000,000,000	30518	

- This is disk space, backups, maintenance, caching, logging, replication and HA feature performance and scalability, etc.

** Taken from Kimberly Tripp's Pluralsight Course:*

SQL Server: Why Physical Database Design Matters

<http://www.pluralsight.com/courses/sqlserver-why-physical-db-design-matters>



T-SQL TIPS

T-SQL HABITS THAT CAN CAUSE PROBLEMS

T-SQL Tips



NOLOCK

Allows a Dirty Read

- Does not issue locks to prevent other transactions from modifying data being read
- Allows other transactions to modify the data while you're trying to read it
- Data returned to the SELECT statement may or may not actually exist in the database, and in some cases it may cause a query to return the same row multiple times or even skip rows

When should I use NOLOCK?

- If your query doesn't necessarily need to return precise figures, and can tolerate some inconsistencies
- If you are querying data that does not get modified often

But NOLOCK makes my query faster!

- It makes your query faster because it is ignoring the safeguards put in place to ensure that your query is returning accurate data

If you need 100% accurate results from your query, do not use NOLOCK

T-SQL Tips

Stored Procedures

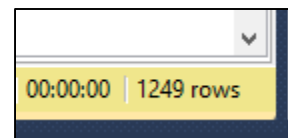

Do not name your stored procedures with the “sp_” prefix!

- This is reserved for system stored procedures
- SQL Server first checks the Master database for these procedures

Use SET NOCOUNT

- Can improve stored procedure performance
- Turns off the messages that SQL Server sends back to the client after each T-SQL statement is executed

```
1 CREATE PROCEDURE <Procedure_Name, sysname, ProcedureName>
2
3 AS
4 BEGIN
5     -- SET NOCOUNT ON added to prevent extra result sets from
6     -- interfering with SELECT statements.
7     SET NOCOUNT ON;
8
9
10 END
11 GO
12
```



00:00:00	1249 rows
----------	-----------

T-SQL Tips



Query Performance Killers

- Try not to use ORDER BY or DISTINCT in your queries, as it adds a lot of extra overhead. It is more efficient to sort/filter the data in your application
- Format Dates in your application instead of in your SELECT statements
- Avoid Cursors and Loops in your T-SQL statements, as it forces row-by-row operations
- Using Functions in WHERE clauses and JOINS should be avoided
- Do Not Use SELECT *
 - Can cause the optimizer to ignore indexes on the table, forcing a full table scan
 - The more fields you return, the worse your performance is going to be (especially when ordering)
- Avoid Data Type Mismatches (aka Implicit Conversions)
 - Variables used in WHERE clauses should match the data type of the columns they're compared with
 - Columns used in JOIN conditions should have matching data types



cf

Thank you!