



SQL SERVER TIPS

FOR EVERYDAY PROGRAMMERS

Who is this guy?

Eric Cobb

- Started in IT in 1999 as a "webmaster"
- Developer for 14 years
- MCSE: Data Platform
- Now a full time DBA

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

Blog Twitter: @sqlnugg

Personal Twitter: @cfgears



What are we going to learn?

- How your database design can impact resources and performance
- Performance tips for your tables, indexes, and stored procedures
- Design tips to incorporate into your database development
- Common T-SQL mistakes and things to avoid
- Primarily focusing on OLTP databases
- Tips that I've picked up from being a DBA



SQL SERVER TIPS

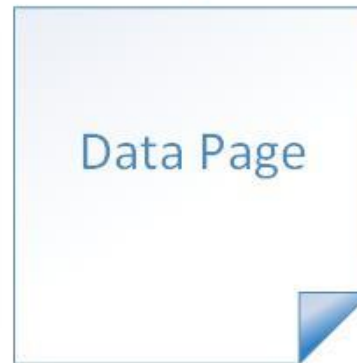
FOR EVERYDAY PROGRAMMERS

A PEEK UNDER THE HOOD OF SQL SERVER

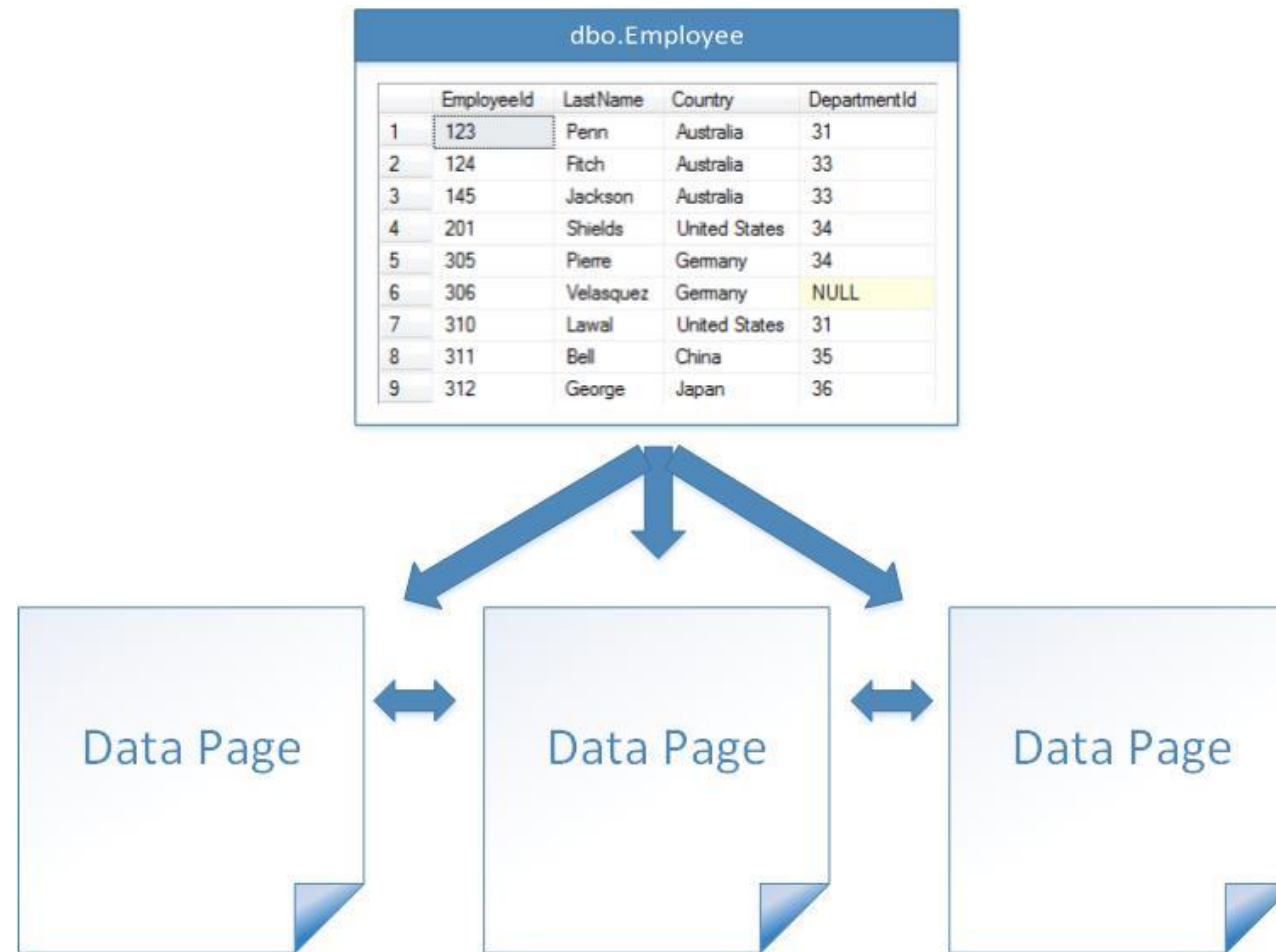
A BRIEF OVERVIEW OF HOW SQL SERVER STORES AND RETRIEVES DATA

A Peek Under The Hood

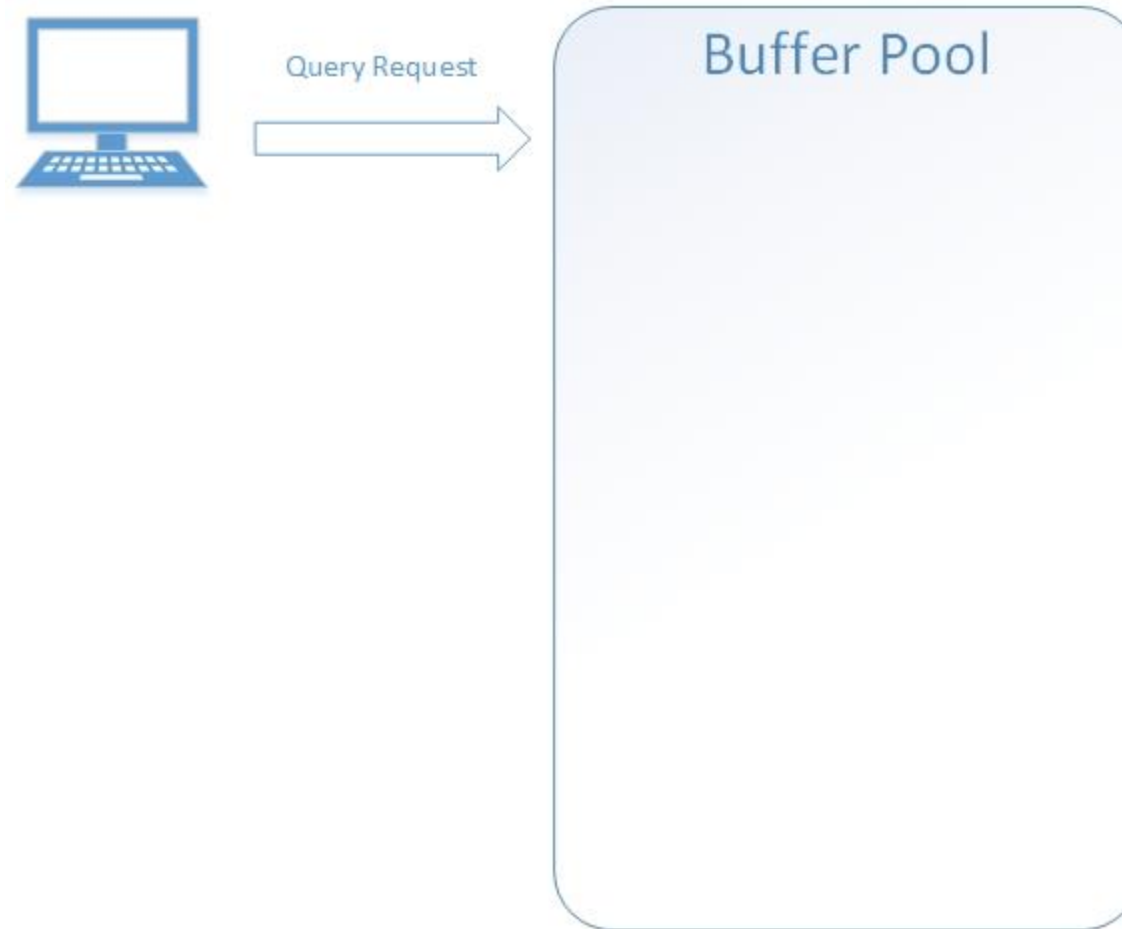
dbo.Employee				
	EmployeeId	LastName	Country	DepartmentId
1	123	Penn	Australia	31
2	124	Fitch	Australia	33
3	145	Jackson	Australia	33
4	201	Shields	United States	34
5	305	Pierre	Germany	34
6	306	Velasquez	Germany	NULL
7	310	Lawal	United States	31
8	311	Bell	China	35
9	312	George	Japan	36



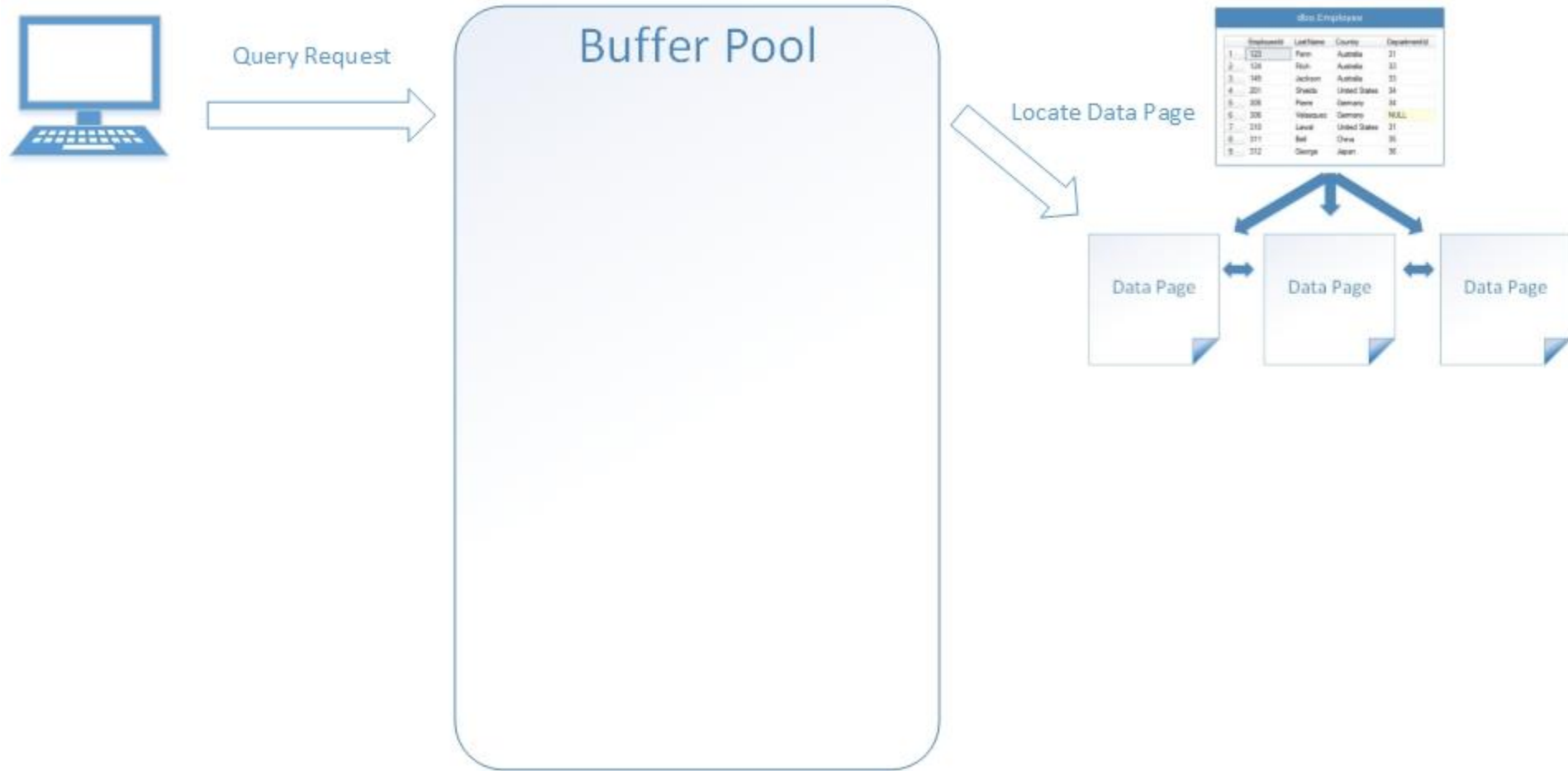
A Peek Under The Hood



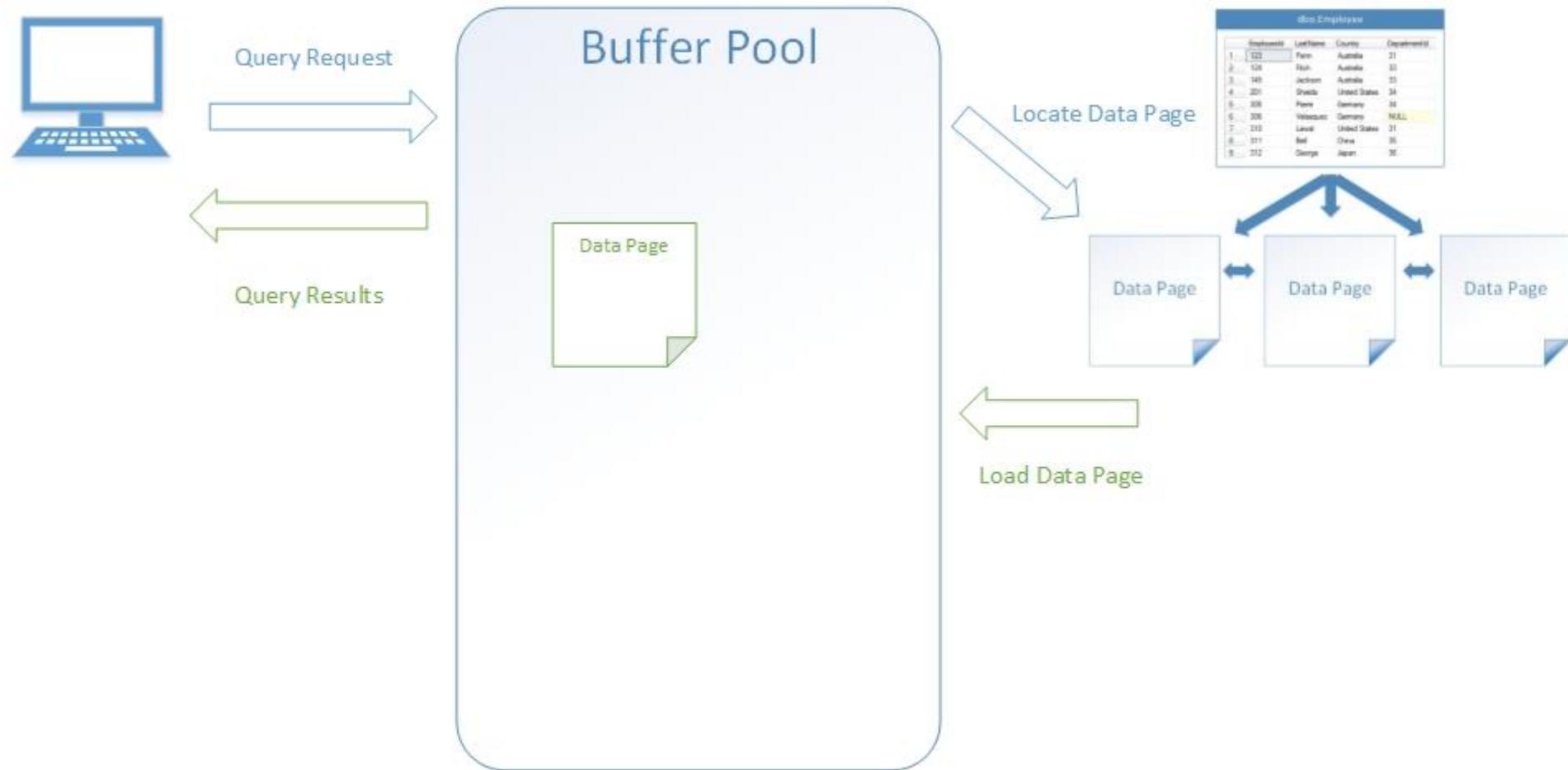
A Peek Under The Hood



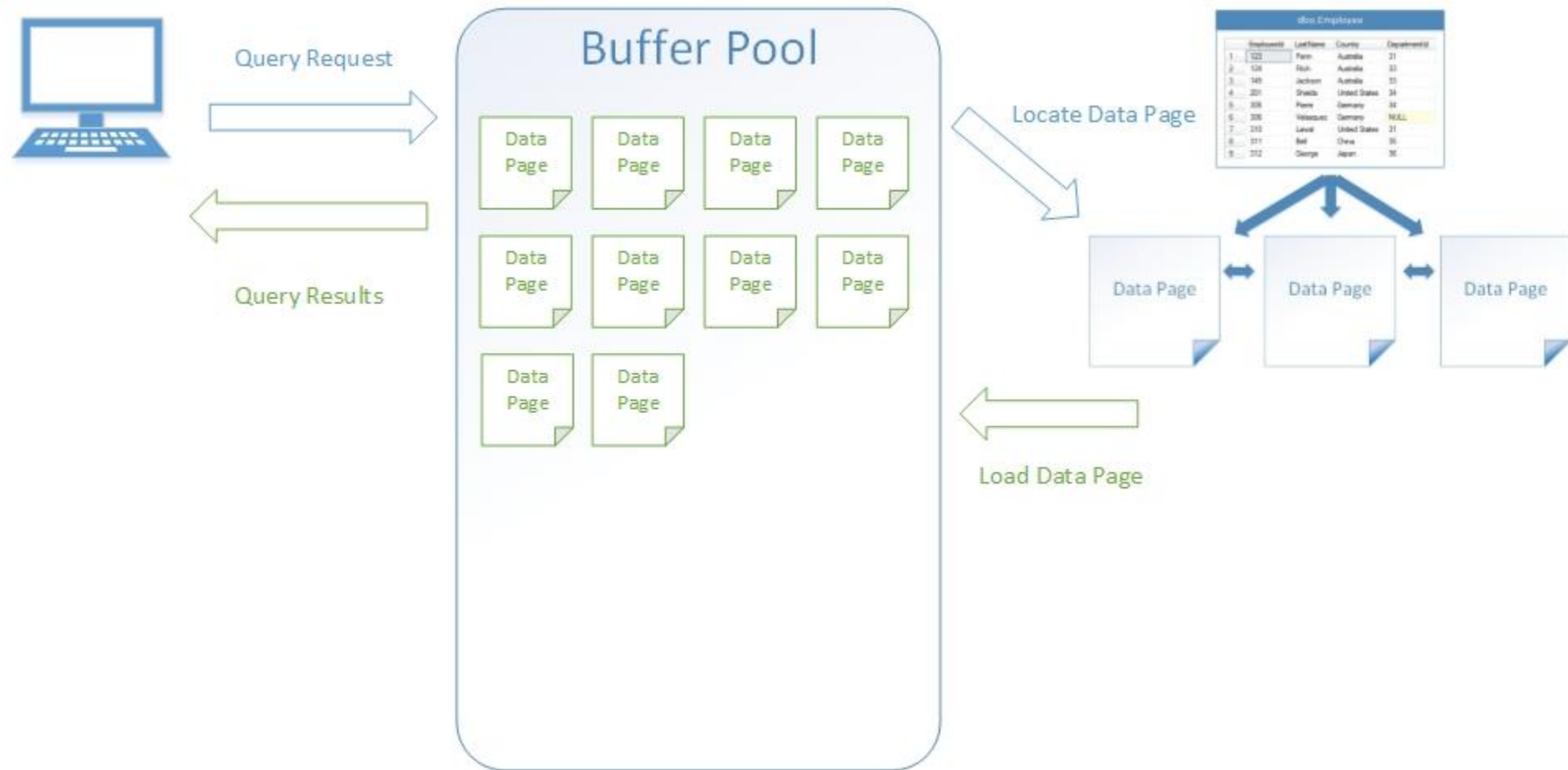
A Peek Under The Hood



A Peek Under The Hood



A Peek Under The Hood



A Peek Under The Hood

Why Does This Matter?

Wasted space in data pages equals:

- Wasted space on the hard drive
- Wasted RAM in the Buffer Pool
- Slower queries
- Larger backups
- Slower backups and restores

More efficient DB design equals more efficient resource usage

- Faster queries
- Better scalability
- Critical when dealing with limited or constrained resources



SQL SERVER TIPS

FOR EVERYDAY PROGRAMMERS

BUILDING BETTER TABLES

DESIGNING TABLES WITH EFFECIENCY IN MIND

Building Better Tables

How is the data ordered in a Table (Data Page)?

Unstructured Data (Heap)

- Data is stored in the heap without specifying an order
- Query optimizer reads all the rows in the table (table scan), and extracts the rows that meet the query criteria
- A table scan generates many disk I/O operations and can be resource intensive
- Should generally be avoided, although can be useful when inserting large amounts of data in ETL/Bulk processes

Structured Data (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 linked for faster sequential access
- 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

Building Better Tables

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 Noppharatchathaniburirom Udomratchaniwetmahasathan Amonphimanawatansathit Sakkathattiyawitsanukamprasit*. (176 chars)

Building Better Tables

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

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

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

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

Why does this matter?

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 takes longer to rebuild an existing application than it does to originally design one correctly.

Building Better Tables

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>



SQL SERVER TIPS

FOR EVERYDAY PROGRAMMERS

UNDERSTANDING INDEXES

A LOOK AT MORE EFFICIENT DATA RETRIEVAL

Understanding Indexes

What Are Indexes?

- Speed retrieval of data from a table
 - Improves performance of SELECT statements
 - Also used in UPDATE and DELETE statements
- Without an index, SQL Server has to check every row in the table
 - This is called a table scan; should be avoided
- Proper indexing is one of the best performance enhancements you can make to your database

Understanding Indexes

Indexes

Clustered Indexes

- Tells SQL Server how to physically sort the records on disk
- The most important index you can apply to a table
- There is only ever 1 clustered index on a table

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

Understanding Indexes

Indexing Tips

Clustered indexes

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

Choose a good Clustered Index (Primary Key) for your table

- Should be unique, narrow, static, and incremental
- Good Clustered Index example:
 - A numeric identity column (smallint, int, bigint)
- Clustered Indexes to avoid:
 - Unique Identifier (GUID/UUID)
 - Character columns (CHAR, VARCHAR, NVARCHAR, etc...)
 - Combination of multiple character columns (LastName, FirstName, MiddleInitial)
 - Columns that undergo frequent changes

Understanding Indexes

Indexes

Nonclustered Indexes

- Used for supporting queries
- Copies the values from the specified columns
- Points to the actual data rows (via Clustered Index or Heap Row ID)
- Created manually
- Can have multiple Nonclustered Indexes on a table
 - SQL Server 2005 supports up to 249 per table
 - SQL Server 2008+ supports up to 999 per table
- Foreign Key != Index
 - SQL Server does NOT automatically create indexes on foreign key columns
 - Indexing foreign keys can provide performance benefits
- Clustered Index is always included with Nonclustered Indexes

Understanding Indexes

Indexing Tips

Nonclustered Indexes

- Create on columns used in your WHERE and JOIN conditions, and columns referenced by IN predicates
- A column with few unique values is seldom a good candidate to be indexed
- Create across multiple columns, Indexes on single columns are rarely useful (except for FK's)
- The order of the columns in the index is important
 - Should be ordered from the most distinct to the least distinct
- Nonclustered Indexes can be filtered for smaller, more targeted result sets
- It is better to have fewer indexes that can serve many queries than it is to have indexes created specifically for each query

Understanding Indexes

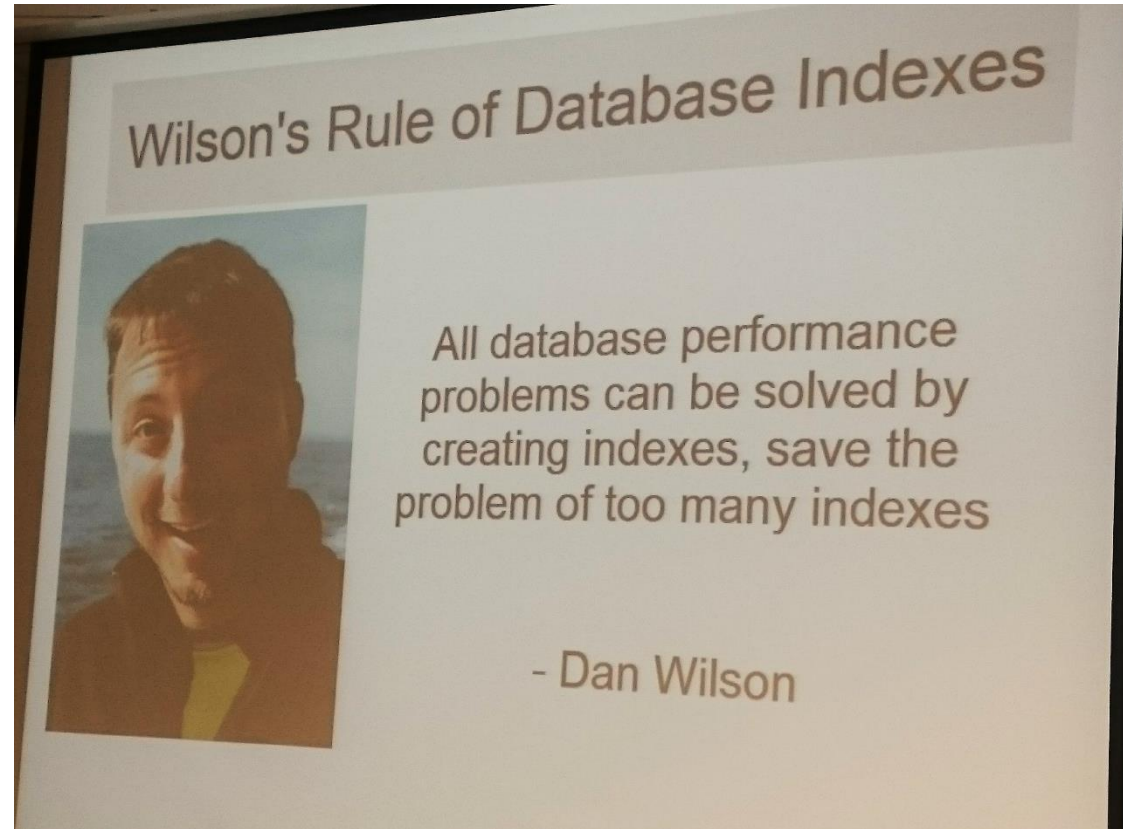
Indexing Considerations

Indexes can both help and hinder performance

- Indexes are written to disk, every index you create will take up space in your database
- OLAP and OLTP have different Index considerations
 - OLAP tables – mainly focused on reads, generally more heavily indexed
 - OLTP tables – mix of reads and writes, needs fewer, more precise indexes
- Only build the indexes you need to help your queries, do not randomly create indexes on every column.
- Indexes are automatically updated when Inserts, Updates, and Deletes are performed on the table
 - More indexes = more processing
- Avoid over-indexing heavily updated tables

Understanding Indexes

Indexing Considerations





SQL SERVER TIPS

FOR EVERYDAY PROGRAMMERS

T-SQL TIPS

A LOOK AT SOME T-SQL HABITS THAT CAN HURT QUERY PERFORMANCE

T-SQL Tips

NOLOCK

Same as setting the Transaction Isolation Level to READ UNCOMMITTED

- 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

Allows a Dirty Read

- 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?

- When your query doesn't necessarily need to return precise figures, and can tolerate some inconsistencies
- If you can deal with "not 100% accurate" results from your query, then NOLOCK may be OK

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
- What is the point in having a faster query if it doesn't return the correct results?

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

T-SQL Tips

Stored Procedures

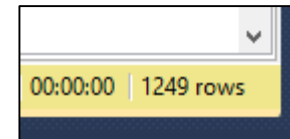

Do not name your stored procedures with the “sp_” prefix!

- 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
```



T-SQL Tips

Do Not Use SELECT *

- Select only the columns you need
- Retrieves more data from the database than you really need, which causes more data to move from the database to the client, and takes more time to travel across the network
- If any columns are added to the table in the future your query will be returning that data as well, possibly causing much larger result sets
- Can cause the optimizer to ignore indexes on the table, forcing a full table scan, which is less efficient than using an index
- When used in Views, SELECT * can return data under the wrong column, or completely skip columns altogether

T-SQL Tips

Query Performance Killers

- Do not use ORDER BY in your SELECT statements unless you really need to, as it adds a lot of extra overhead. If possible, it's more efficient to sort the data in your application.
- Try not to use function calls on table columns in SELECT statements or the WHERE clause
 - WHERE Year(orderDate) = 2016
- Avoid Cursors and Loops in your T-SQL. Forces row-by-row operations
- Make sure your data types match in your queries
 - 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

Questions?



Thank you!

Eric Cobb

<http://www.sqlnuggets.com>