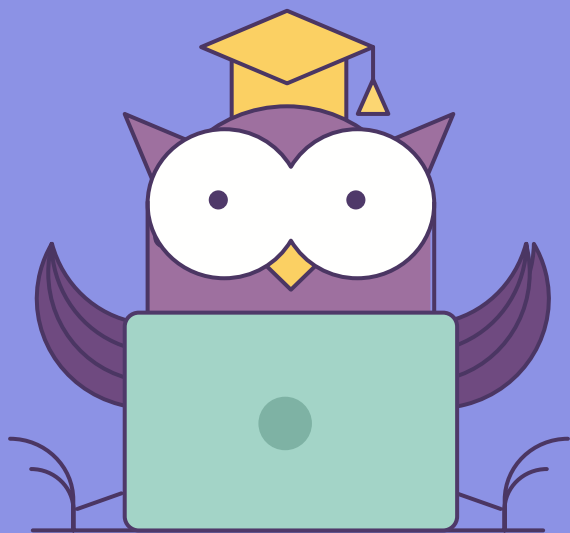





ОНЛАЙН-ОБРАЗОВАНИЕ

# Меня хорошо слышно && видно?



Напишите в чат, если есть проблемы!

Ставьте  если все хорошо  
Или напишите, какие есть проблемы

Не забыть включить  
запись!



1. Когда полезна связь “один-к-одному”
2. Надо хранить только дату. Какой тип использовать?  
Date, Time, Datetime или Datetime2?
3. Есть ли разница между типами DateTime и Datetime2?
4. Что такое “Collation”? Для каких объектов задается?

# Индексы

Курс “MS SQL Server разработчик”  
Группа 2021-03



- Что это такое “индекс” и для чего нужен?
- Какие виды индексов вы знаете?



**Индекс (англ. index)** — объект базы данных, создаваемый с целью повышения производительности поиска данных.

- Таблицы в базе данных могут иметь большое количество строк, которые хранятся **в произвольном порядке**, и их **поиск** по заданному критерию путём последовательного просмотра таблицы строка за строкой может занимать **много времени**.
- Индекс формируется из значений **одного или нескольких столбцов** таблицы.
- Ускорение работы с использованием индексов достигается в первую очередь за счёт того, что **индекс имеет структуру, оптимизированную под поиск** — например, сбалансированного дерева.

# Что такое индекс

## Index

### Symbols

1NF (first normal form)  
2NF (second normal form)  
\* (asterisk)  
performance, 4  
SELECT lists of  
\ (backslash), named instances, 14  
[<Character>-<Character>] (Character List or Range)  
, (comma), 37, 265  
{ } curly brackets, set theory, 4  
" (double quotes), 64  
@@@identity function  
[<List of Characters>]  
@params, 360  
() parentheses  
column aliases  
derived tables, 161  
functions, 80  
precedence, 52  
% (percent) wildcard  
+ (plus sign) operator  
; (semicolon)  
MERGE, 272  
statements, 21  
' (single quotes), 64  
.sql script files, 385  
@stmt, 360  
\_ (underscore) wildcard

### A

ABC flavors, 12  
access, views using

ANSI SQL-92 syntax  
cross joins, 100  
inner joins, 103  
appliance flavor, 12  
APPLY operator, 178–181, 306  
arguments  
CTEs, 165  
derived tables, 161  
arithmetic operators, 51  
arrays, 1NF, 8  
AS, inline aliasing, 160  
assignment SELECT, 340  
assignment UPDATE, 269  
asterisk (\*)  
performance, 41  
SELECT lists of subqueries, 139  
atomicity, attributes, 7  
attributes  
atomicity, 7  
blocking\_session\_id attribute, 308  
expressions, 36  
filtering in outer joins, 115  
foreign key constraints, 23  
nullability, 20  
set theory, 4  
autonumbering, assignment UPDATE, 269

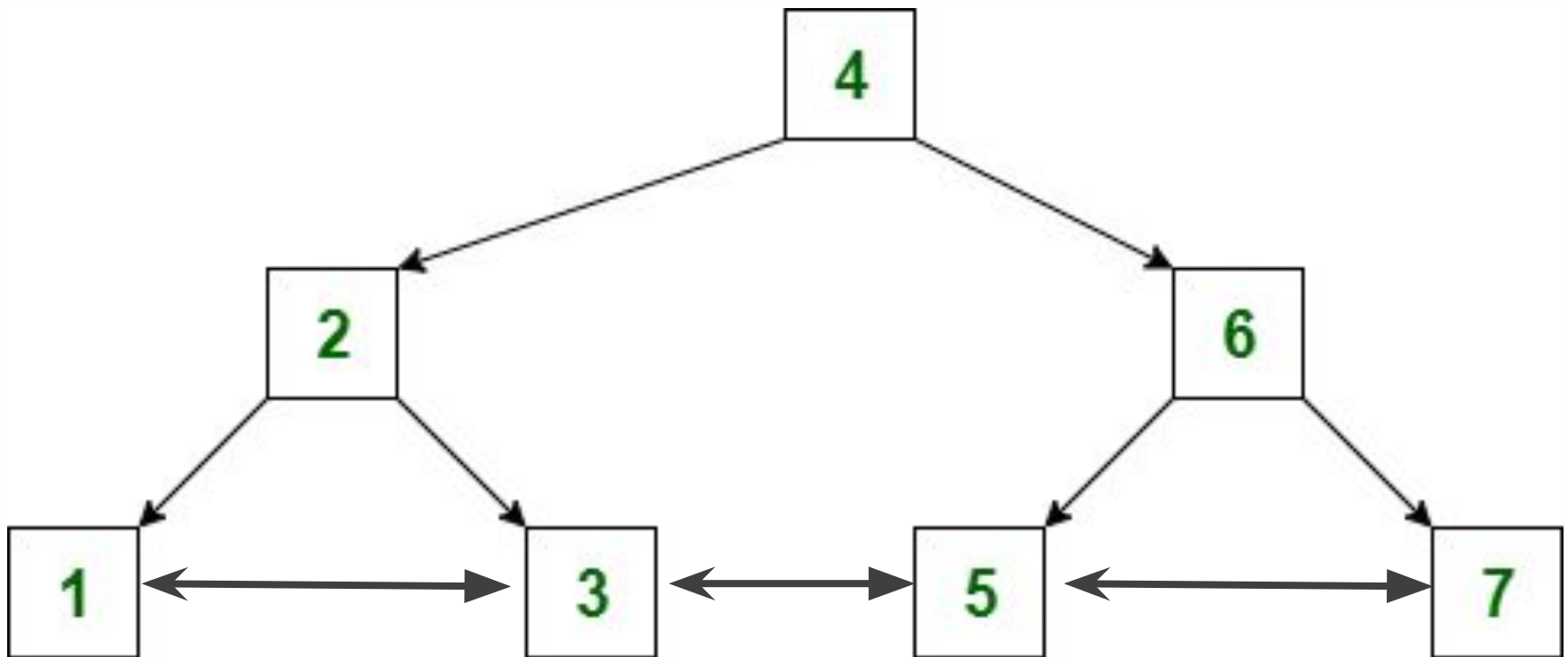
### B

backslash (\), named instances, 14  
bag, 3  
batches, 341–345  
GO, 344  
statements that cannot be combined in the same batch, 343  
as a unit of parsing, 342  
as a unit of resolution, 344  
variables, 343  
BEGIN, 346  
BEGIN TRAN, 297  
BETWEEN, 50  
BISM (Business Intelligence Semantic Model), 11  
blockers, terminating, 308  
blocking. See locks and blocking  
blocking\_session\_id attribute, 308  
boundaries, transactions, 297  
box flavor, 13  
BULK INSERT, 252  
Business Intelligence Semantic Model (BISM), 11

### C

caching, sequence objects, 257  
candidate keys  
3NF, 8  
about, 7  
Cantor, George, set theory, 3  
Cartesian products  
cross joins, 99  
inner joins, 103  
CASE expressions  
about, 53  
pivoting data, 225  
CAST function, 81, 138  
catalog views, 88  
CATCH blocks, 371  
character data, 61–73  
collation, 62  
data types, 61  
LIKE predicate, 71  
operators and functions, 64–71  
character data types, 51  
CHARINDEX function, 67  
check constraints, 24  
CHECK, @@identity and SCOPE\_IDENTITY, 255  
CHECK OPTION option, 174  
CHOOSE function, 55  
clauses, defined, 29  
close world assumption (CWA), 5  
cloud flavor, 13  
COALESCE function, 66  
Codd, Edgar F., relational model, 4  
coding style, 21  
collation  
character data, 62  
property, 16  
COLUMNPROPERTY function, 90  
columns  
aliases  
assigning, 159  
CTEs, 164  
query example, 38  
referencing within a SELECT clause, 42  
asterisk in column names, 41  
attributes in set theory, 4  
external column aliasing, 169  
identity property, 255  
INSERT VALUES, 248  
ordinal position  
in SQL, 41  
in T-SQL, 43





- **Концептуальный уровень**

- Модель предметной области без привязки к модели данных.
- Сущности, связи, атрибуты.

- **Логический уровень**

- Представление данных с точки зрения выбранной модели данных (реляционная или другая)
- Реляционная модель: отношения, кортежи, атрибуты, нормализация/денормализация и тд

- **Физический уровень**

- Представление данных с точки зрения выбранной СУБД
- Таблицы, колонки и тп.
- Где находятся файлы БД, в каких файлах какие таблицы
- **Индексы**
- и др

- Полнотекстовые индексы
- Колоночные индексы (Column store)
- XML индексы
- Пространственные индексы
- "Обычные" B+-tree
  - Кластерные, не кластерные
  - Составные индексы
  - Покрывающие (INCLUDE)
  - Фильтрованные (WHERE)

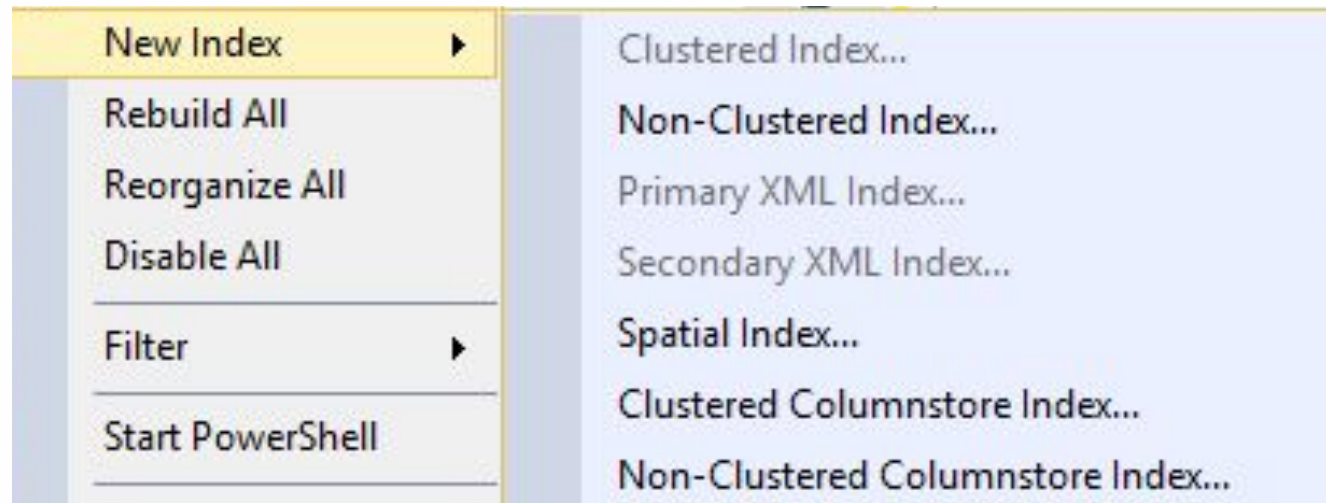


## не сегодня

- Статистика
- Анализ использования индексов
- Rebuild и reorg индексов в системе
- Структура индекса, fillfactor, как они хранятся

Это еще будет впереди

- “Обычные” (B+, кластерные, не кластерные)
- Полнотекстовые (full text)
- Колоночные (column store)
- Пространственные (spatial)
- XML



# 01

## Полнотекстовые индексы

“Полнотекстовые запросы выполняют лингвистический поиск в текстовых данных в полнотекстовых индексах путем обработки слов и фраз в соответствии с правилами конкретного языка.” (из [документации](#))

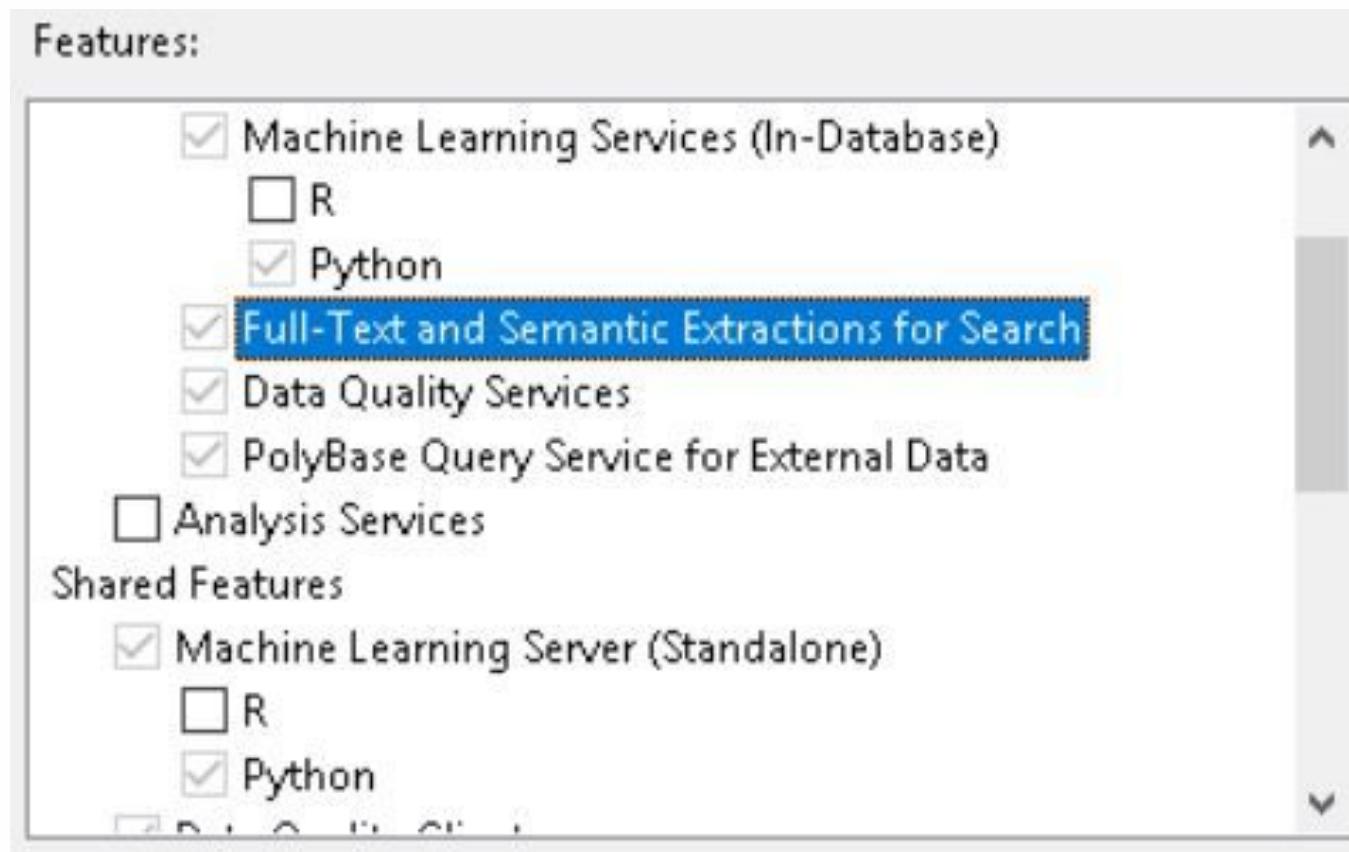
- Поиск “как в google”
- С SQL Server 7.0
- В char-полях
- И в бинарных (doc, pdf, tiff и др)
- Аналоги функционала: Elasticsearch, Sphinx, Apache Lucene

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16  
*Now is the time for all good men to come to the aid of the party*

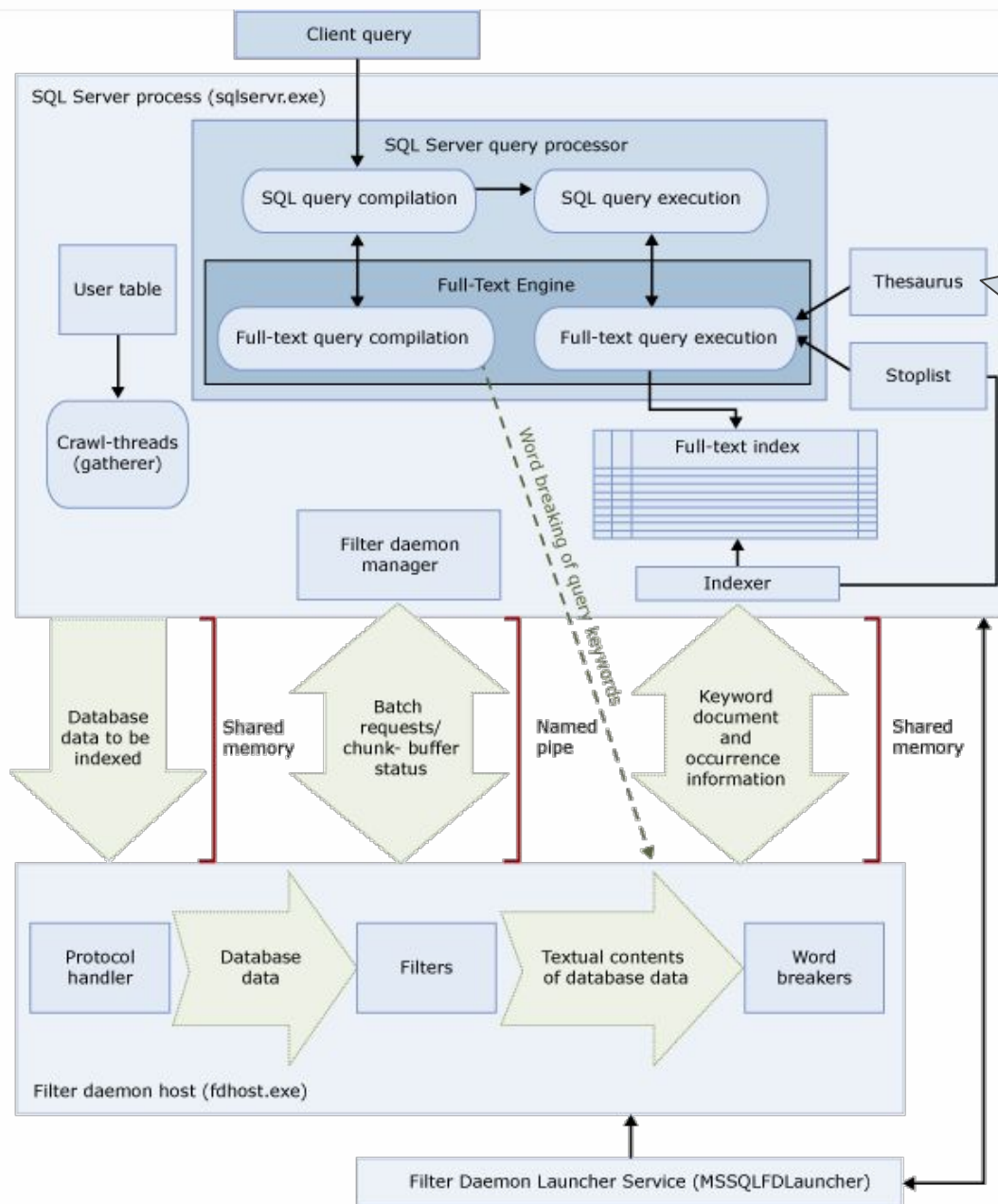
Word	Document ID	Occurrence
time	1	4
good	1	7
men	1	8
aid	1	13
party	1	16
{End Of File}	1	17

**Figure 1-2.** *Inverted index with stopwords removed*

Отдельный компонент при установке SQL Server







## Синонимы

C:\Program Files\Microsoft SQL Server\MSSQL14.SQL 2017\MSSQL\FTData

- Создать полнотекстовый каталог  
*CREATE FULLTEXT CATALOG*
- Убедиться, что есть уникальный индекс  
*CREATE UNIQUE INDEX*
- Создать полнотекстовый индекс  
*CREATE FULLTEXT INDEX*
- Наполнить индекс  
*ALTER FULLTEXT INDEX ON ...  
START FULL POPULATION;*

- CONTAINS/CONTAINSTABLE

- поиск точных и неточных соответствий отдельных слов и фраз.
- указывать уровень сходства похожих слов;
- возвращать взвешенные совпадения;
- объединять условия поиска с логическими операторами.

- FREETEXT/FREETEXTTABLE

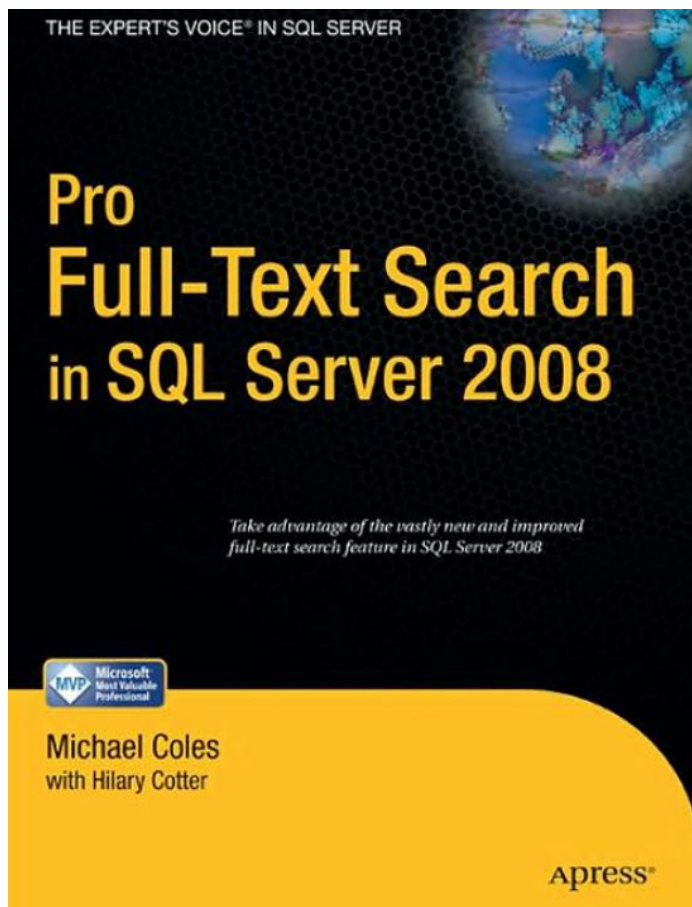
- Поиск совпадений по смыслу, а не по буквальному совпадению задаваемых слов, фраз или предложений (текст в свободной форме).
- Соответствие регистрируется, если в полнотекстовом индексе указанного столбца найден любой из терминов в любой форме.

<https://docs.microsoft.com/ru-ru/sql/relational-databases/search/query-with-full-text-search>

# ДЕМО

## Полнотекстовый поиск





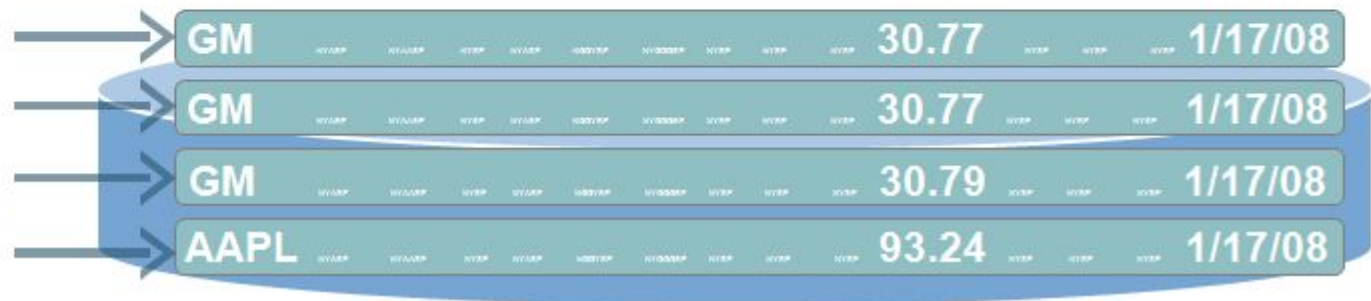
- Michael Coles with Hilary Cotter  
**Pro Full-Text Search in SQL Server 2008**
- [Документация SQL Server](#)

# 02

## Колоночные индексы (Column store)

## Хранение по строкам

Чтение всех столбцов



## Хранение по столбцам

Чтение 3х столбцов



Колоночные СУБД — принцип действия, преимущества и область применения (habr)

Примеры колоночных СУБД:

Vertica, Greenplum, SAP HANA, MonetDB, ClickHouse

**Column Store** – данные хранятся в виде сжатых сегментов, каждый сегмент содержит часть строк одной колонки

- Аналитические запросы



Подробнее — [Колоночные индексы в SQL Server 2016 \(youtube\)](#)



- Впервые появились в SQL Server 2012 (с многими ограничениями), потом развивались и улучшились.  
Нормально можно пользоваться с SQL Server 2016
- Для OLAP, агрегирования данных.
- Высокая степень сжатия
- Таблицы с большим количеством столбцов - читаем только необходимые
- Кластерные (с 2014) и не кластерные (с 2012)
- CREATE CLUSTERED COLUMNSTORE INDEX
- CREATE [NONCLUSTERED] COLUMNSTORE INDEX

# ДЕМО

## Column Store



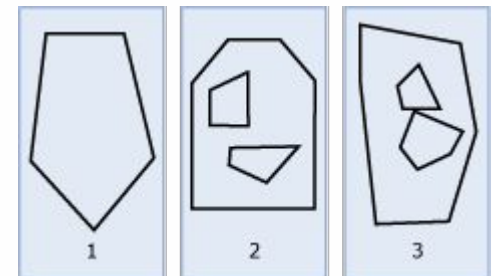
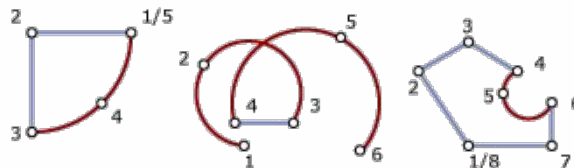
Differences	NCCI Не кластерный	CCI Кластерный
Index Columns	Need to specify a list of columns to create NCCI on. A typical usage is to only include columns needed for analytics. If underlying table has an unsupported column, for example, a spatial datatype or LOB column, you can exclude it	All columns are included. For example, if the underlying table has an unsupported column, for example LOB, you will be required to vertically partition the table. Note, SQL Server team is working to support large datatypes post SQL Server 2016
Storage	Creating NCCI does not save any storage. In fact, it can potentially add approximately 10% additional storage as it is a new index. In many customer cases, once you create an NCCI, you may be able to drop one or more btree nonclustered indices which were created primarily for querying large set of rows.	Creating CCI can compress the data 10x. Of course, the compression achieved will depend upon the schema and the data but 10x is what you can expect. If table was originally PAGE compressed, you can expect around 5x compression
Workload	NCCI is designed to be used for workloads where you have a mix of transactional and analytics workload. In fact, NCCI is the preferred index for real-time operational analytics in SQL Server 2016	CCI is designed for DW workloads. A typical implementation will have FACT tables with CCI and Dimension tables with rowstore. However, there are no hard-and-fast rules but to get benefit of CCI or NCCI, the number of rows must exceed 1 million rows but preferably much higher
Updatable	Yes (starting with SQL Server 2016)	Yes

Columnstore option	Recommendations for when to use	Compression
Clustered columnstore index	Use for: 1) Traditional data warehouse workload with a star or snowflake schema 2) Internet of Things (IOT) workloads that insert large volumes of data with minimal updates and deletes.	Average of 10x
Nonclustered B-tree indexes on a clustered columnstore index	Use to: 1. Enforce primary key and foreign key constraints on a clustered columnstore index. 2. Speed up queries that search for specific values or small ranges of values. 3. Speed up updates and deletes of specific rows.	10x on average plus some additional storage for the NCI.
Nonclustered columnstore index on a disk-based heap or B-tree index	Use for: 1) An OLTP workload that has some analytics queries. You can drop B-tree indexes created for analytics and replace them with one nonclustered columnstore index. 2) Many traditional OLTP workloads that perform Extract Transform and Load (ETL) operations to move data to a separate data warehouse. You can eliminate ETL and a separate data warehouse by creating a nonclustered columnstore index on some of the OLTP tables.	NCCI is an additional index that requires 10% more storage on average.
Columnstore index on an in-memory table	Same recommendations as nonclustered columnstore index on a disk-based table, except the base table is an in-memory table.	Columnstore index is an additional index.

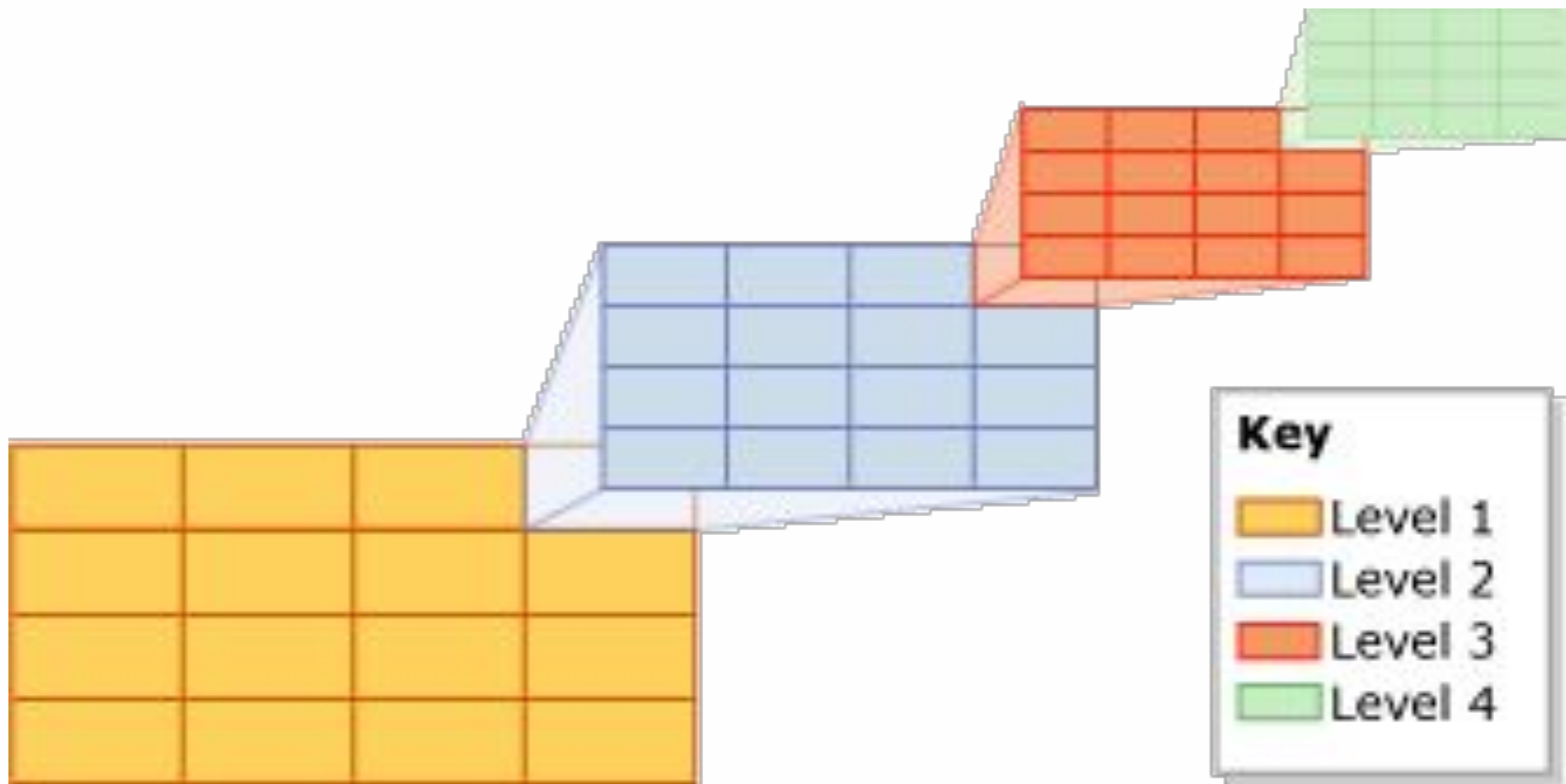
# 03

## Пространственные индексы

- GEOMETRY
  - представляет данные в евклидовой (плоской) системе координат, 2D
- GEOGRAPHY
  - представляет данные в геодезической (географической) системе координат, 3D
- C SQL Server 2008
- CLR-типы



- Использует B-Tree



<https://docs.microsoft.com/ru-RU/sql/relational-databases/spatial/spatial-indexes-overview>

# ДЕМО

## Пространственные индексы





# Делаем перерыв 2-3 мин?

Напишите в чат:

- "+" — нужен перерыв
- "-" — перерыв не нужен



# 04

## XML индексы

- Для XQuery-запросов, когда запрашивается часть xml-данных
- Отличаются от “обычных” индексов
  - Первичный  
индексируются все теги, значения и пути
  - Вторичный  
path, value, property
  - Селективный (выборочный) (с SQL Server 2012)  
аналог фильтрованных индексов (where)
- Должен быть кластерный индекс

```
<BOOK ISBN="1-55860-438-3">
  <SECTION>
    <TITLE>Bad Bugs</TITLE>
    Nobody loves bad bugs.
    <FIGURE CAPTION="Sample bug"/>
  </SECTION>
  <SECTION>
    <TITLE>Tree Frogs</TITLE>
    All right-thinking people
    <BOLD> love </BOLD> tree frogs.
  </SECTION>
</BOOK>
```

Figure 1. Sample XML data

- ❖ Hierarchical dot-separated labels assigned to nodes
  - Compressed binary form used internally
- ❖ Positive odd integers assigned initially
- ❖ Negative, even integers used for insertions

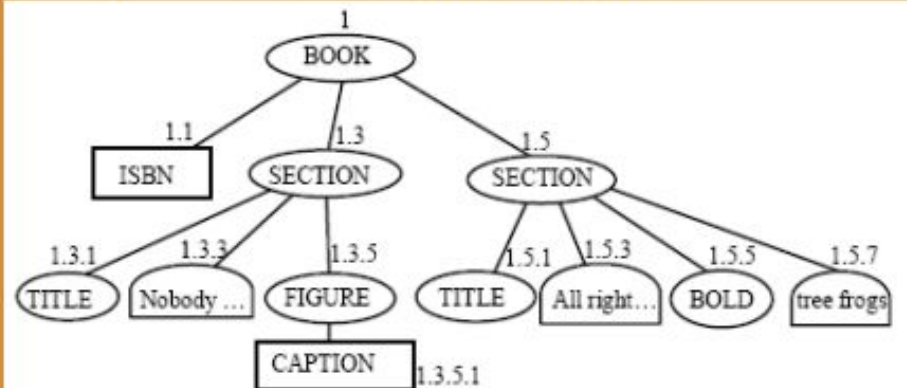


Figure 2. ORDPATH Node Label

XML парсится и представляется в виде таблицы

## ❖ Store "infoset" items of XML nodes in B+ tree

Primary key of XML instance's row in base table (used for back join)

ID	ORDPATH	TAG	NODE_TYPE	VALUE	PATH_ID
1	1	1(BOOK)	1(Element)	Null	#1
	1.1	2(ISBN)	2(Attribute)	'1-55860-438-3'	#2#1
	1.3	3(SECTION)	1	Null	#3#1
	1.3.1	4(TITLE)	1	'Bad Bugs'	#4#3#1
	1.3.3	10(TEXT)	4(Value)	'Nobody loves Bad bugs.'	#10#3#1
	1.3.5	5(FIGURE)	1	Null	#5#3#1
	1.3.5.1	6(CAPTION)	2	'Sample bug'	#6#3#1
	1.5	3(SECTION)	1	Null	#3#1
	<BOOK ISBN="1-55860-438-3">			'Tree frogs'	#4#3#1
	<SECTION>			'All right-thinking people'	#10#3#1
	<TITLE>Bad Bugs</TITLE>			'love'	#7#3#1
	Nobody loves bad bugs.			'tree frogs'	#10#3#1
	<FIGURE CAPTION="Sample bug"/>				
	</SECTION>				
	<SECTION>				
	<TITLE>Tree Frogs</TITLE>				
	All right-thinking people				
	<BOLD> love </BOLD> tree frogs.				
	</SECTION>				
	</BOOK>				

Путь обратном порядке.  
Для запросов вида:  
//SECTION/TITLE

PATH\_VALUE

3	4			2	1
ID	ORDPATH	TAG	NODE_TYPE	VALUE	PATH_ID
1	1	1(BOOK)	1(Element)	Null	#1
	1.1	2(ISBN)	2(Attribute)	'1-55860-438-3'	#2#1
	1.3	3(SECTION)	1	Null	#3#1
	1.3.1	4(TITLE)	1	'Bad Bugs'	#4#3#1
	1.3.3	10(TEXT)	4(Value)	'Nobody loves Bad bugs.'	#10#3#1
	1.3.5	5(FIGURE)	1	Null	
	1.3.5.1	6(CAPTION)	2	'Sample bu	
	1.5	3(SECTION)	1	Null	
	1.5.1	4(TITLE)	1	'Tree frogs'	
	1.5.3	10(TEXT)	4	'All right-th	
	1.5.5	7(BOLD)	1	'love'	
	1.5.7	10(TEXT)	4	'tree frogs'	

Primary key of XML instance's row in base table (used for back join)

<BOOK ISBN="1-55860-438-3">  
 <SECTION>  
     <TITLE>Bad Bugs</TITLE>  
     Nobody loves bad bugs.  
     <FIGURE CAPTION="Sample bug"/>  
 </SECTION>  
 <SECTION>  
     <TITLE>Tree Frogs</TITLE>  
     All right-thinking people  
     <BOLD> love </BOLD> tree frogs.  
 </SECTION>  
</BOOK>

- ❖ Helps with searches for a path + value match (e.g. /BOOK/SECTION[TITLE="Tree frogs"])

Queries that use path expressions, such as the **exists()** XQuery method

Известен путь.



VALUE

3
4
1
2

ID	ORDPATH	TAG	NODE_TYPE	VALUE	PATH_ID
1	1	1(BOOK)	1(Element)	Null	#1
	1.1	2(ISBN)	2(Attribute)	'1-55860-438-3'	#2#1
	1.3	3(SECTION)	1	Null	#3#1
	1.3.1	4(TITLE)	1	'Bad Bugs'	#4#3#1
	1.3.3	10(TEXT)	4(Value)	'Nobody loves Bad bugs.'	#10#3#1
	1.3.5	5(FIGURE)	1	Null	
	1.3.5.1	6(CAPTION)	2	'Sample bug'	
	1.5	3(SECTION)	1	Null	
	1.5.1	4(TITLE)	1	'Tree frogs'	
	1.5.3	10(TEXT)	4	'All right-thinking people	
	1.5.5	7(BOLD)	1	'love'	
	1.5.7	10(TEXT)	4	'tree frogs'	

Primary key of XML instance's row in base table (used for back join)

<BOOK ISBN="1-55860-438-3">  
   <SECTION>  
     <TITLE>Bad Bugs</TITLE>  
     Nobody loves bad bugs.  
     <FIGURE CAPTION="Sample bug"/>  
   </SECTION>  
   <SECTION>  
     <TITLE>Tree Frogs</TITLE>  
     All right-thinking people  
     <BOLD> love </BOLD> tree frogs.  
   </SECTION>  
 </BOOK>

- ❖ Helps when we're looking for a data value but have a wildcard in the path (e.g. /BOOK/SECTION[FIGURE/@\*="Sample Bug"])

Reverse of PATH. Queries that search for values, without knowing the name of the XML element or attribute that contains the value being searched for.

Путь не известен, известно значение.

## Secondary - PROPERTY

PROPERTY

1 ID	4 ORDPATH	TAG	NODE_TYPE	3 VALUE	2 PATH_ID
1	1	1(BOOK)	1(Element)	Null	#1
	1.1	2(ISBN)	2(Attribute)	'1-55860-438-3'	#2#1
	1.3	3(SECTION)	1	Null	#3#1
	1.3.1	4(TITLE)	1	'Bad Bugs'	#4#3#1
	1.3.3	10(TEXT)	4(Value)	'Nobody loves Bad bugs.'	#10#3#1
	1.3.5	5(FIGURE)	1	Null	#5#3#1
	1.3.5.1	6(CAPTION)	2	'Sample bug'	#6#3#1
	1.5	3(SECTION)	1	Null	#3#1

Primary key of XML instance's row in base table

```
SELECT CatalogDescription.value('(/ProductDescription/@ProductModelID)[1]', 'int'),
       CatalogDescription.value('(/ProductDescription/@ProductModelName)[1]', 'varchar(30)')
FROM Production.ProductModel
WHERE ProductModelID = 19
```

- ❖ Helps find (possibly multi-valued) properties of object with known ID and PATH\_ID

This type of index performs very well if the query is trying to retrieve nodes from multiple tuples of the column.

Путь или значение не известны.



Ниже приведены некоторые рекомендации по созданию вторичных индексов.

- Если при работе с XML-столбцами часто используются выражения пути, вторичный XML-индекс PATH, скорее всего, ускорит обработку данных. Типичный пример — выполнение метода `exist()` для XML-столбцов в предложении WHERE инструкции Transact-SQL.
- Если с использованием выражений пути извлекаются множественные значения из отдельных экземпляров XML, может принести пользу кластеризация путей в пределах каждого экземпляра XML в индекс PROPERTY. Этот сценарий обычно имеет место при работе с наборами свойств, когда извлекаются свойства объекта и известно значение его первичного ключа.
- Если запрашиваются значения экземпляров XML, не зная имен элементов или атрибутов, содержащих эти значения, следует подумать о создании индекса VALUE. Как правило, это имеет место при уточняющем запросе по осям нижних уровней, например `//author[last-name="Howard"]`, где элементы `<author>` могут встречаться на любом уровне иерархии. Кроме того, такая ситуация встречается при обработке запросов с символами-шаблонами (например, `/book [@* = "novel"]`, где в запросе выполняется поиск элементов `<book>`, имеющих некоторый атрибут со значением "novel").

[Документация SQL Server — XML Indexes](#)

- [Документация SQL Server — XML Indexes](#)
- Статья про внутреннее устройство XML-индексов  
["Indexing XML Data Stored in a Relational Database"](#)  
([презентация](#))
- [Getting Started With XML Indexes](#)

# ДЕМО

## XML-индексы




# 05


## “Обычные” индексы


- Кластерные, не кластерные
- Составные индексы (несколько колонок)
- Покрывающие (INCLUDE)
- Фильтрованные (WHERE)

- Поиск (WHERE)
- Сортировка (ORDER BY)
- JOIN
- Группировка (GROUP BY), MIN, MAX
- и не только

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ]  
INDEX index_name ON <object> ( column [ ASC |  
DESC ] [ ,...n ] )  
    [ INCLUDE ( column_name [ ,...n ] ) ]  
    [ WHERE <filter_predicate> ]  
    [ ON filegroup_name ]
```

 IX\_Application\_People\_FullName

 PK\_Application\_Countries

 UQ\_Application\_Countries\_CountryName

# 05.1

## Кластерные и некластерные индексы



## Кластерные индексы

- Это и есть таблица
- Может быть только один
- Рекомендации по кластерному ключу
  - Чем меньше размер тем лучше
  - Уникальный (но можно создать и неуникальный)
  - Неизменяемый
  - Постоянно увеличивающийся
  - NOT NULL
  - Фиксированной длины
  - "GUID or not GUID" (UUID) поговорим позже

*PK vs CI — см. пример в Sales.CustomerTransactions*

- *PK — CustomerTransactionID*
- *CI — TransactionDate*

## Не кластерные индексы

- Храняться отдельно от данных таблицы
- Может быть несколько (до 999)

# Heap

Heap – Куча – таблица без кластерного индекса, физический порядок хранения данных в таблице (то есть порядок расположения данных на диске не задан).

id	index_id = 0	first_iam_page	
----	--------------	----------------	--

IAM – index allocation map pages

Отдельные строки идентифицируются по ссылке на идентификатор строки (RID)

RID состоит из номера файла, номера страницы данных и слота на странице

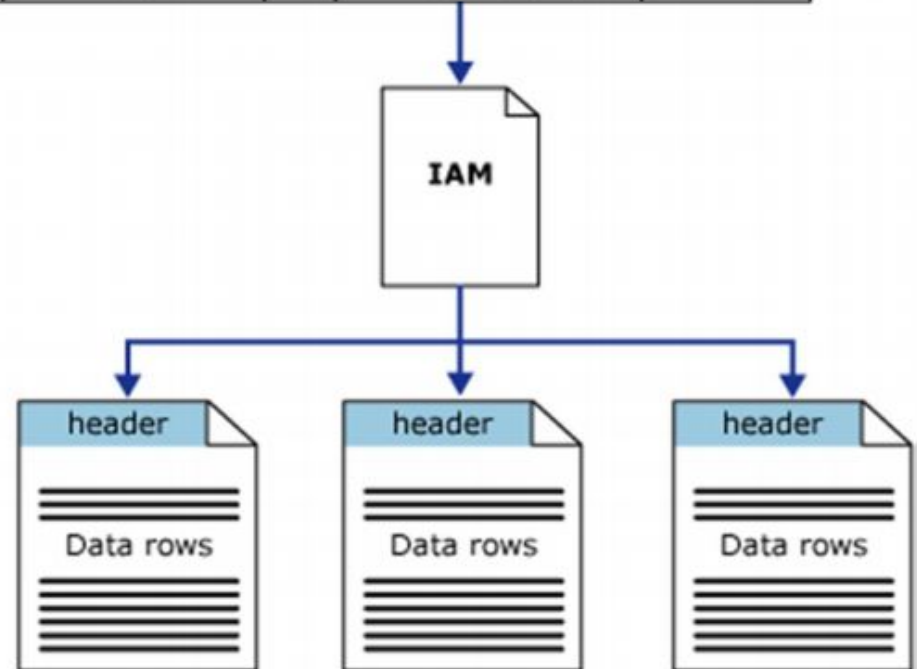


Table Scan



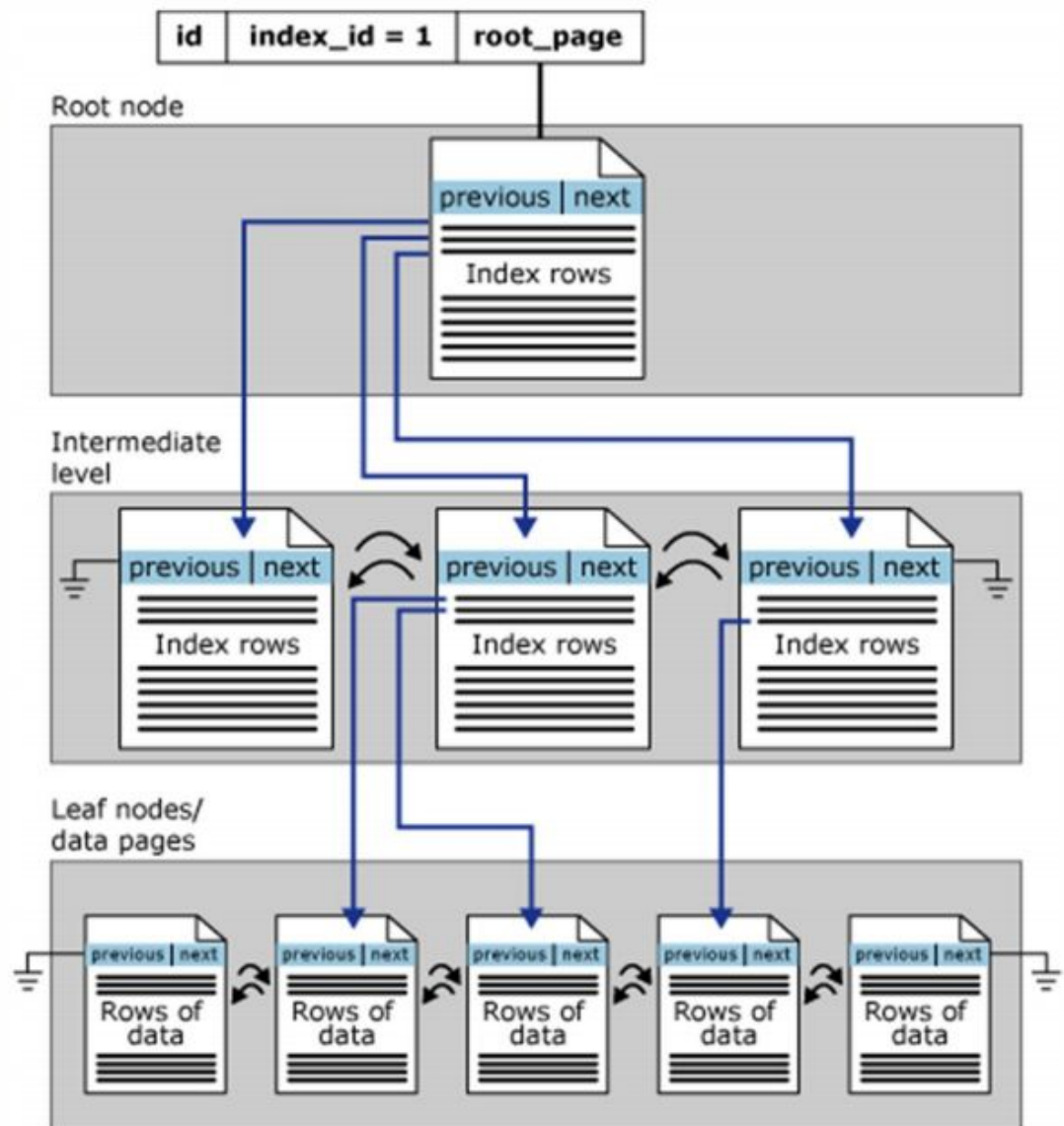
RID Lookup (Heap)

# Clustered

**Clustered table** – таблица с кластерным индексом.

Кластерный индекс – задает порядок расположения физических данных на диске.

Некластерные индексы содержат ссылку на кластерный индекс (само поле (поля) кластерного индекса).



# 05.2

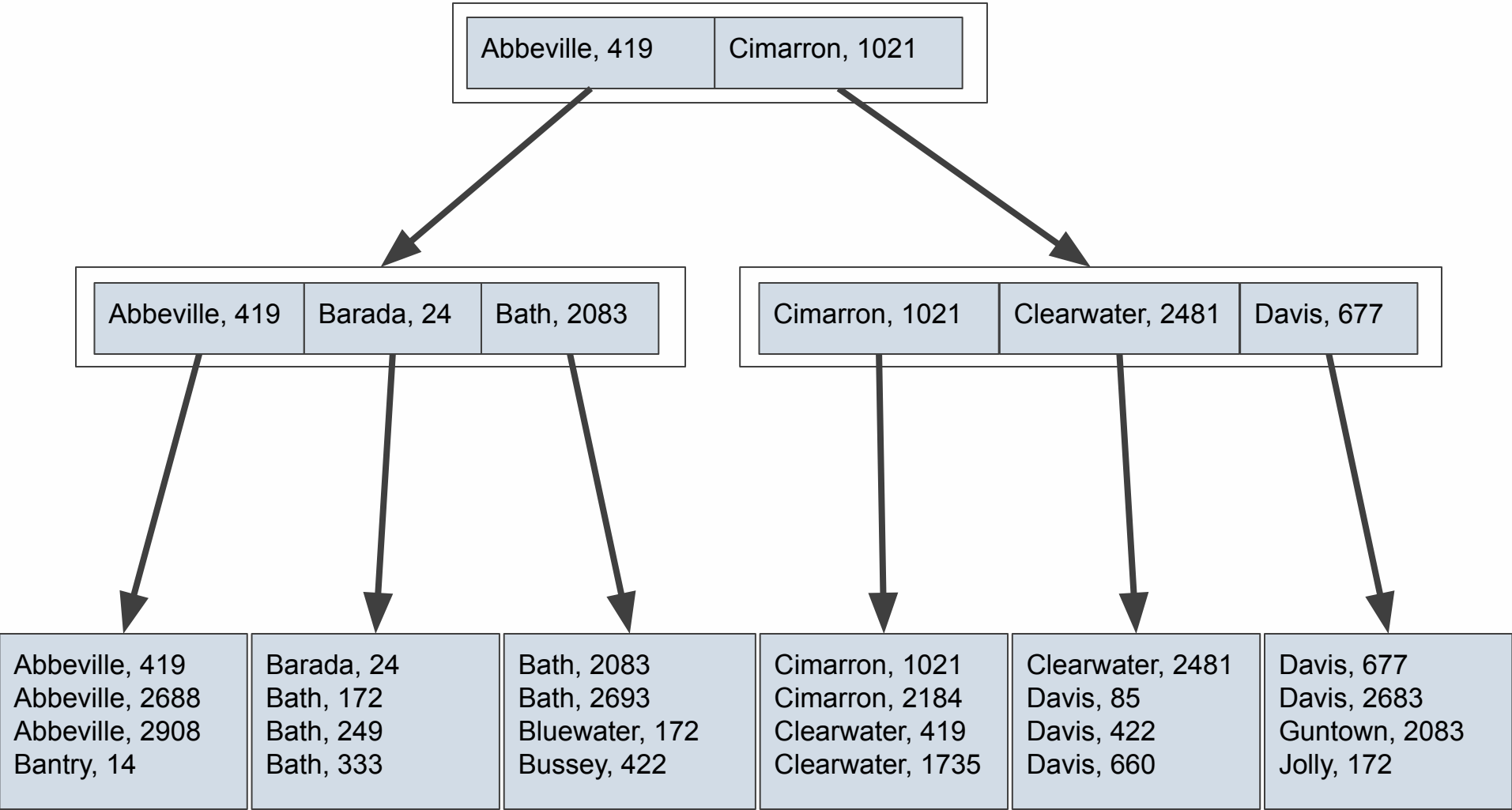
## Составные индексы

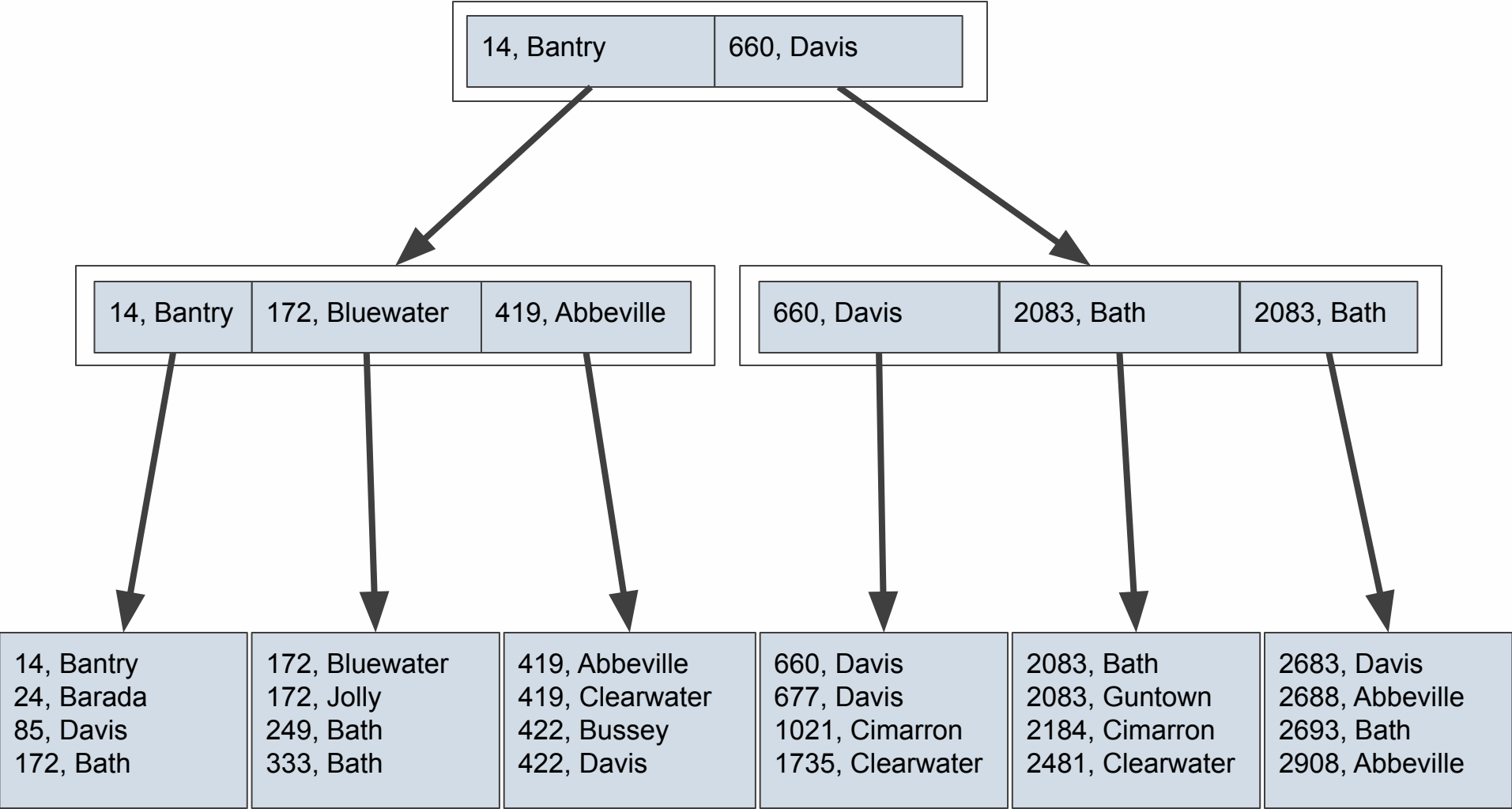
- Несколько столбцов в индексе
  - До 32 столбцов
  - Общая длина ограничена 900 байтами (кластерный), 1700 байт (некластерный)
  - Может быть как кластеризованный, так и некластеризованный
- 
- Важен ли порядок столбцов в индексе?
  - Важен ли порядок столбцов в where?

# Составной индекс

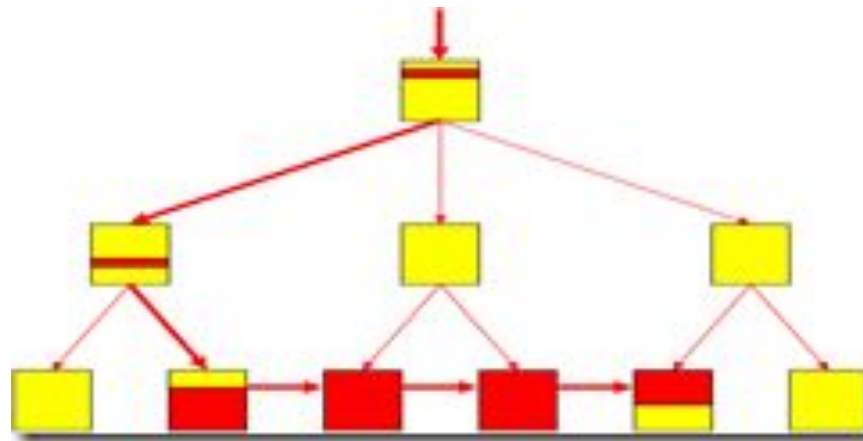
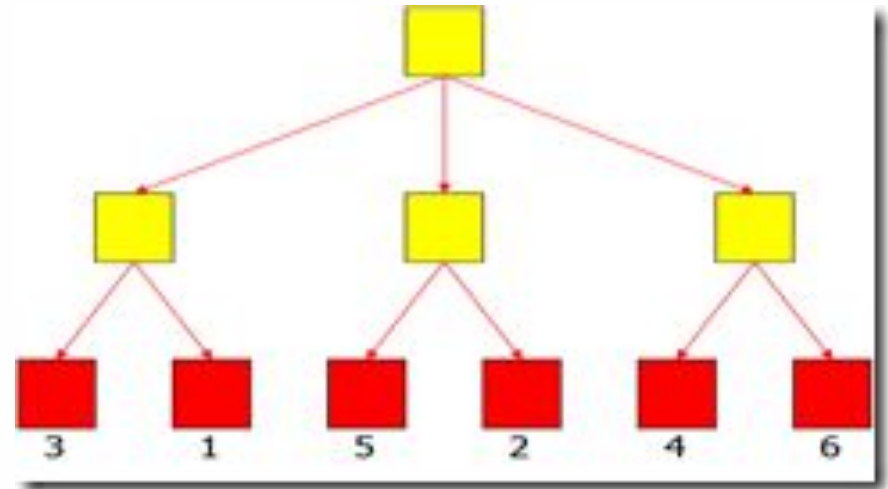
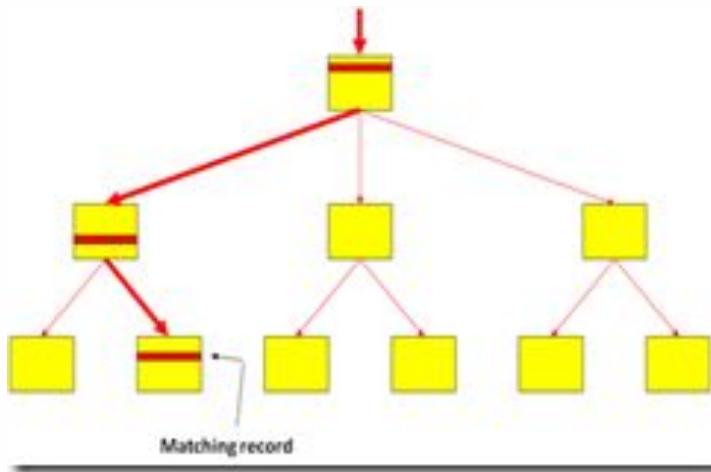
CityID	CityName	StateProvinceID	LatestRecordedPopulation
5	Abbeville	11	2908
6	Abbeville	1	2688
8	Abbeville	25	419
1900	Bantry	35	14
1905	Barada	28	24
2101	Bath	23	2083
2102	Bath	43	172
8551	Davis	43	85
14206	Guntown	25	2083
17143	Jolly	45	172
...	...	...	...

- IX\_Name\_Population
- IX\_Population\_Name





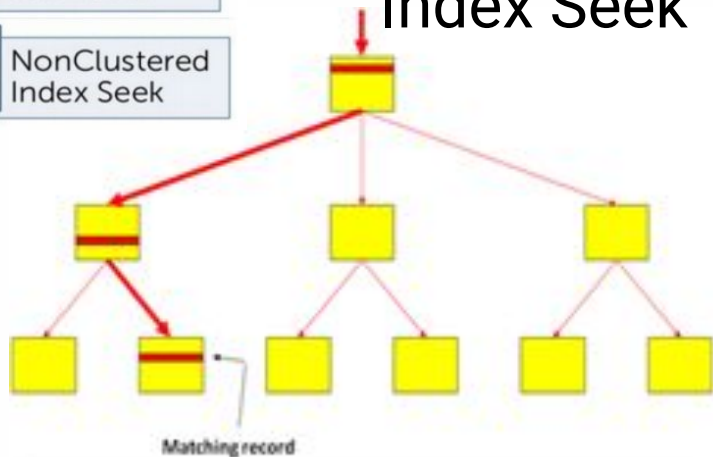




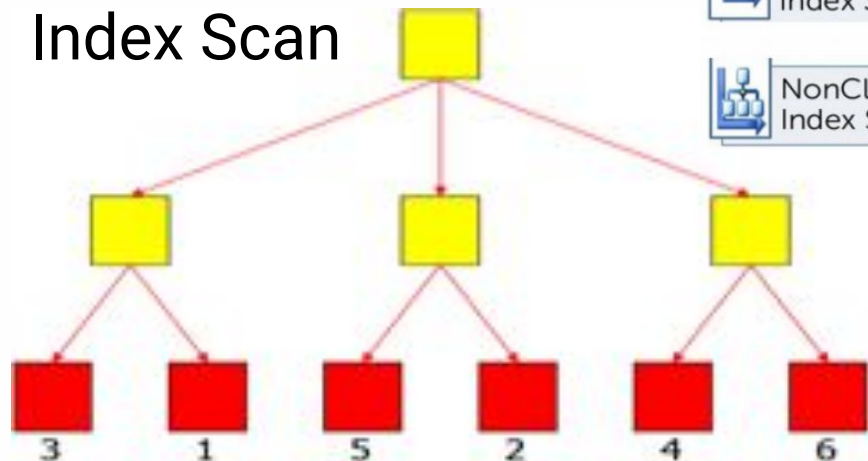
# Вспомним как можно читать индекс?



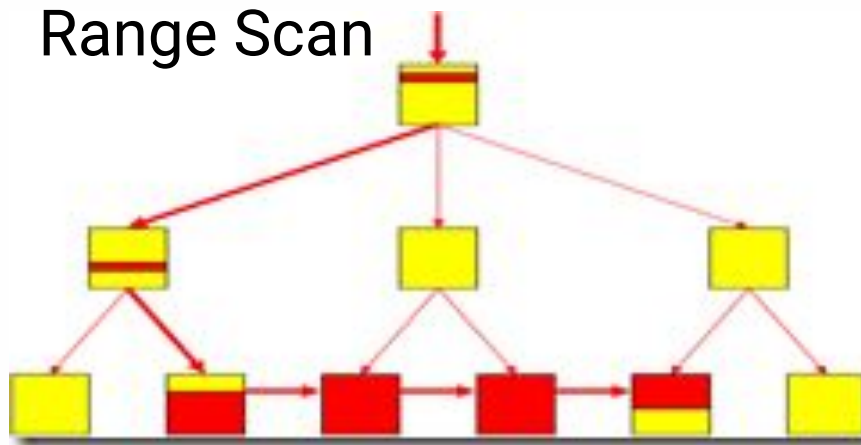
## Index Seek



## Index Scan

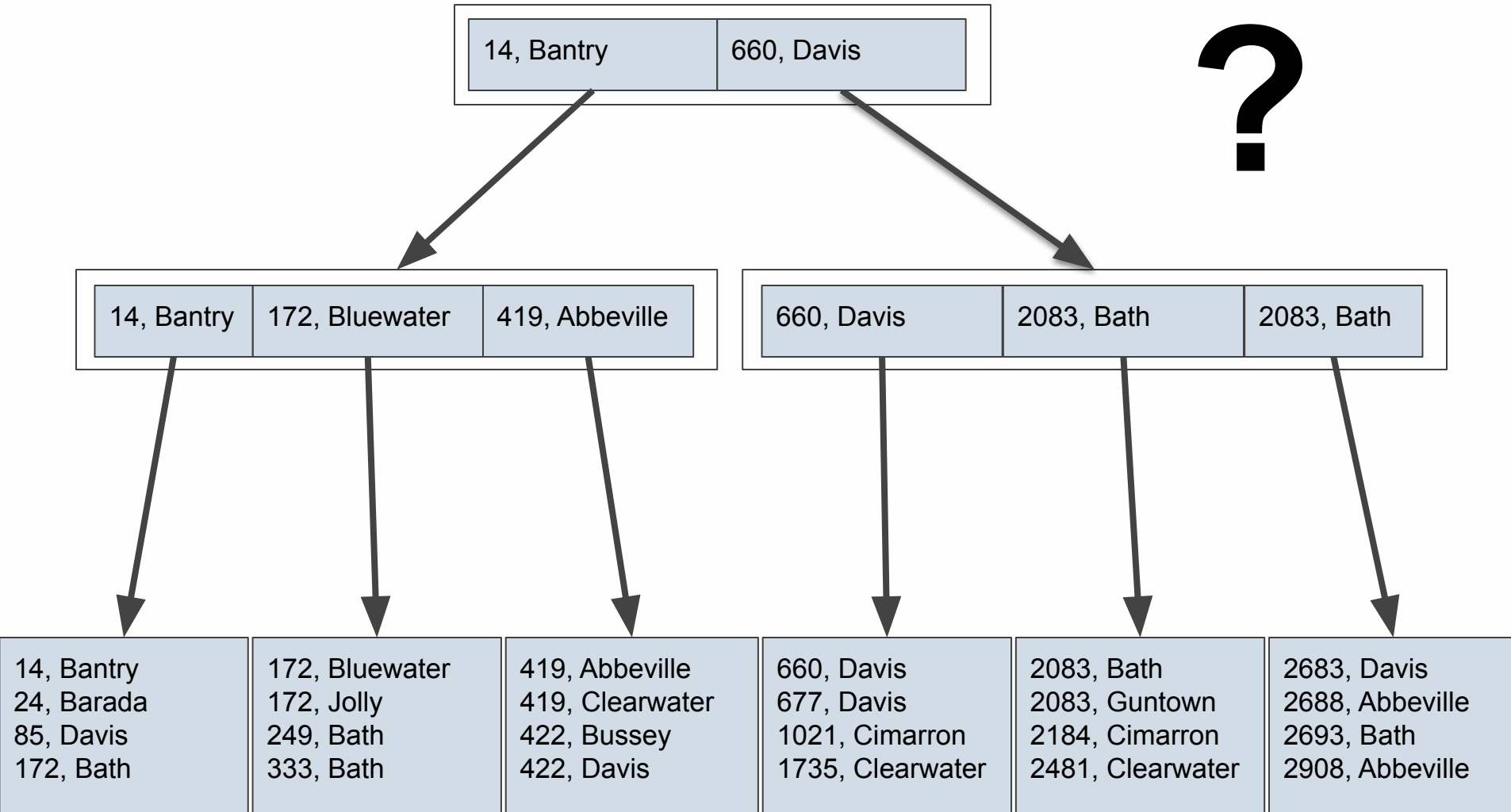


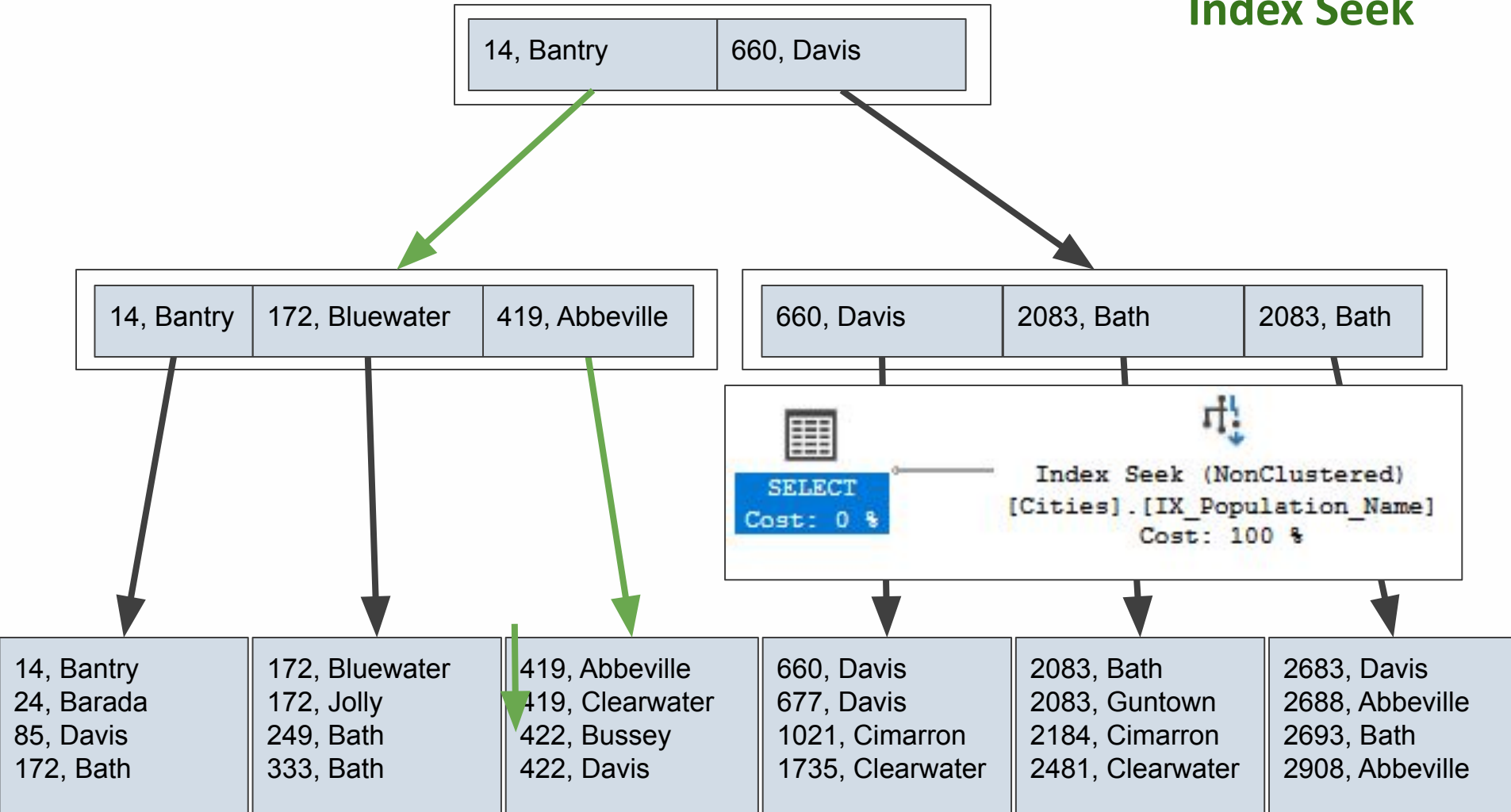
## Range Scan



**WHERE Population = 422**

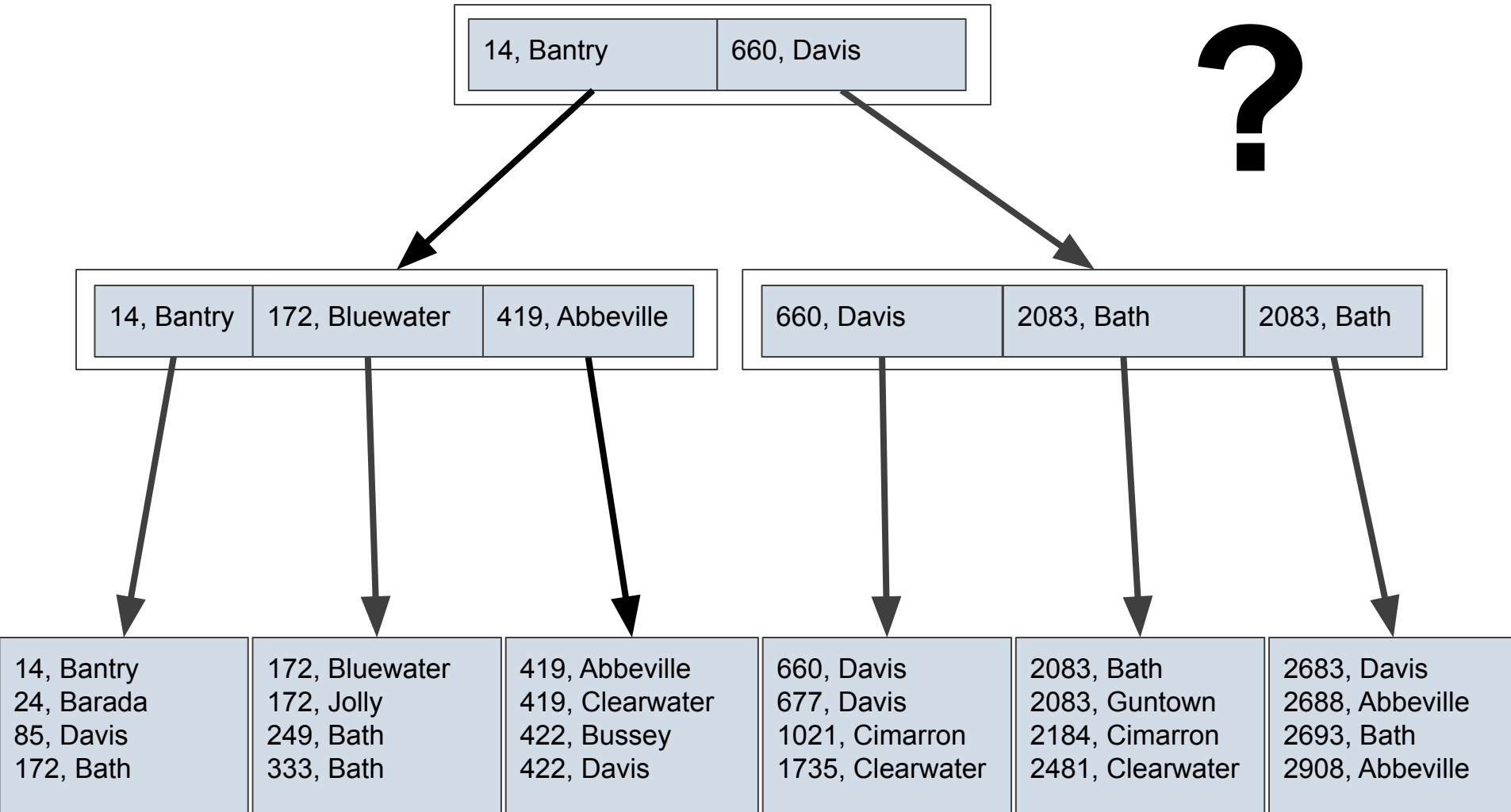
?

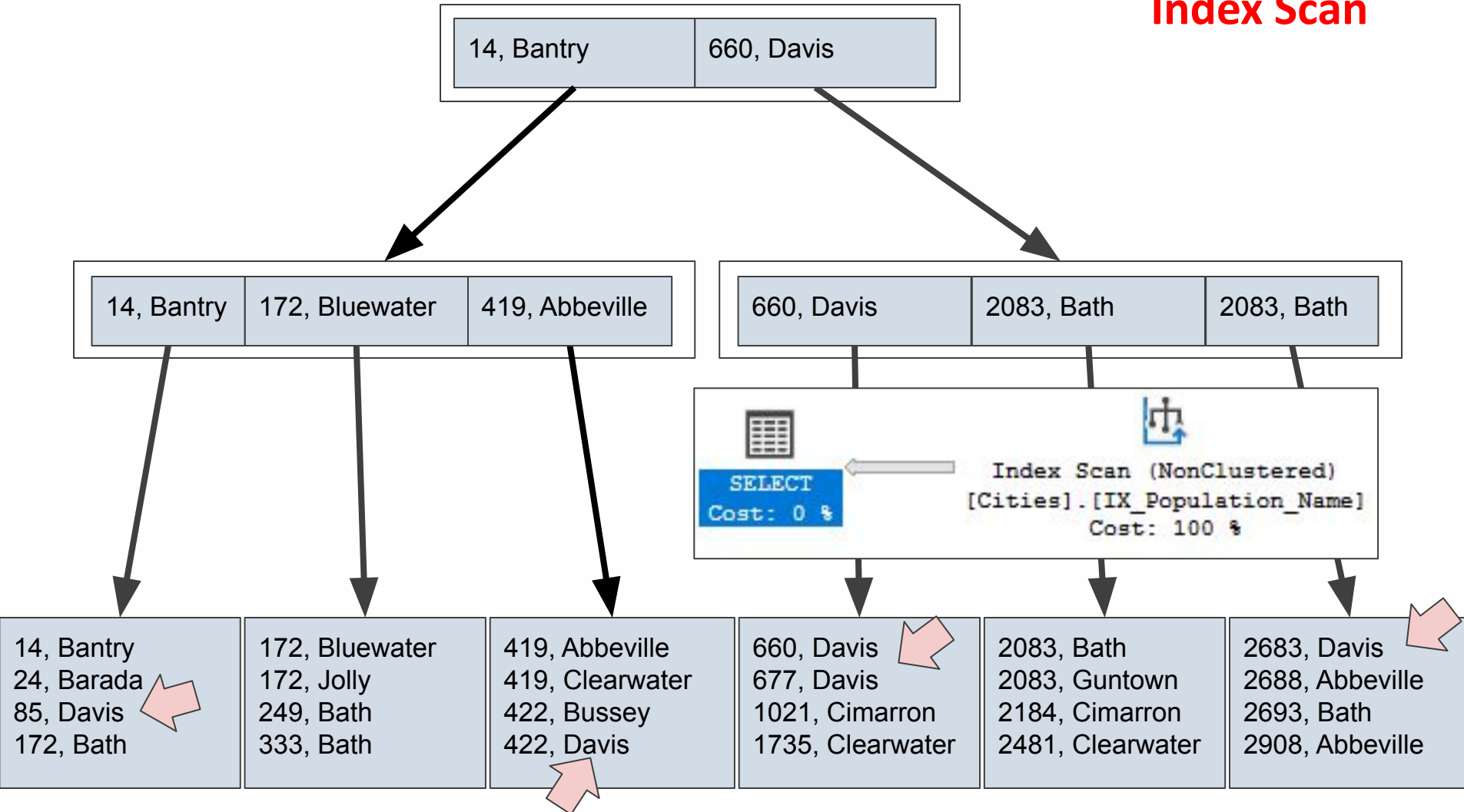


**WHERE Population = 422****Index Seek**

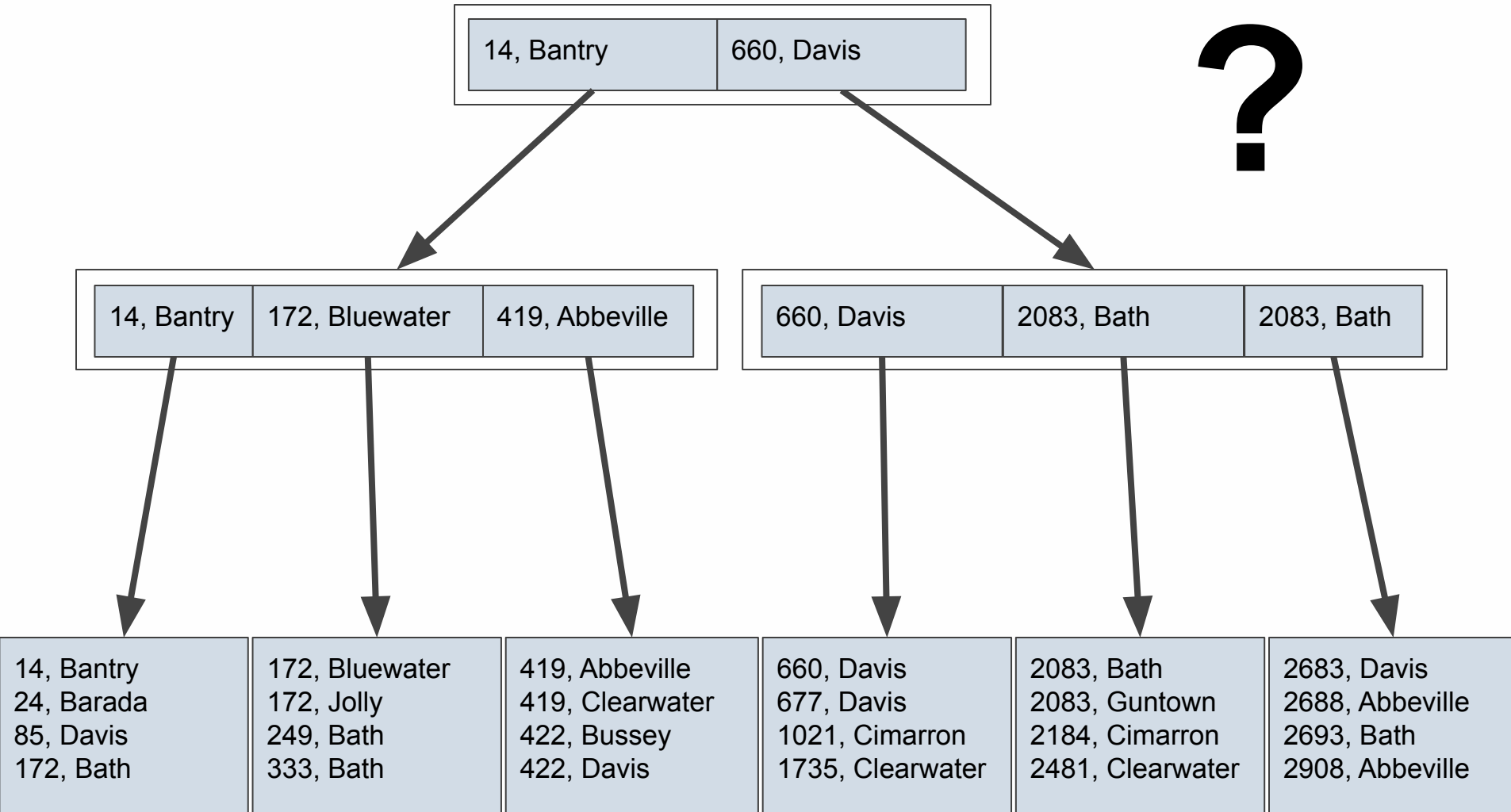
WHERE CityName = 'Davis'

?



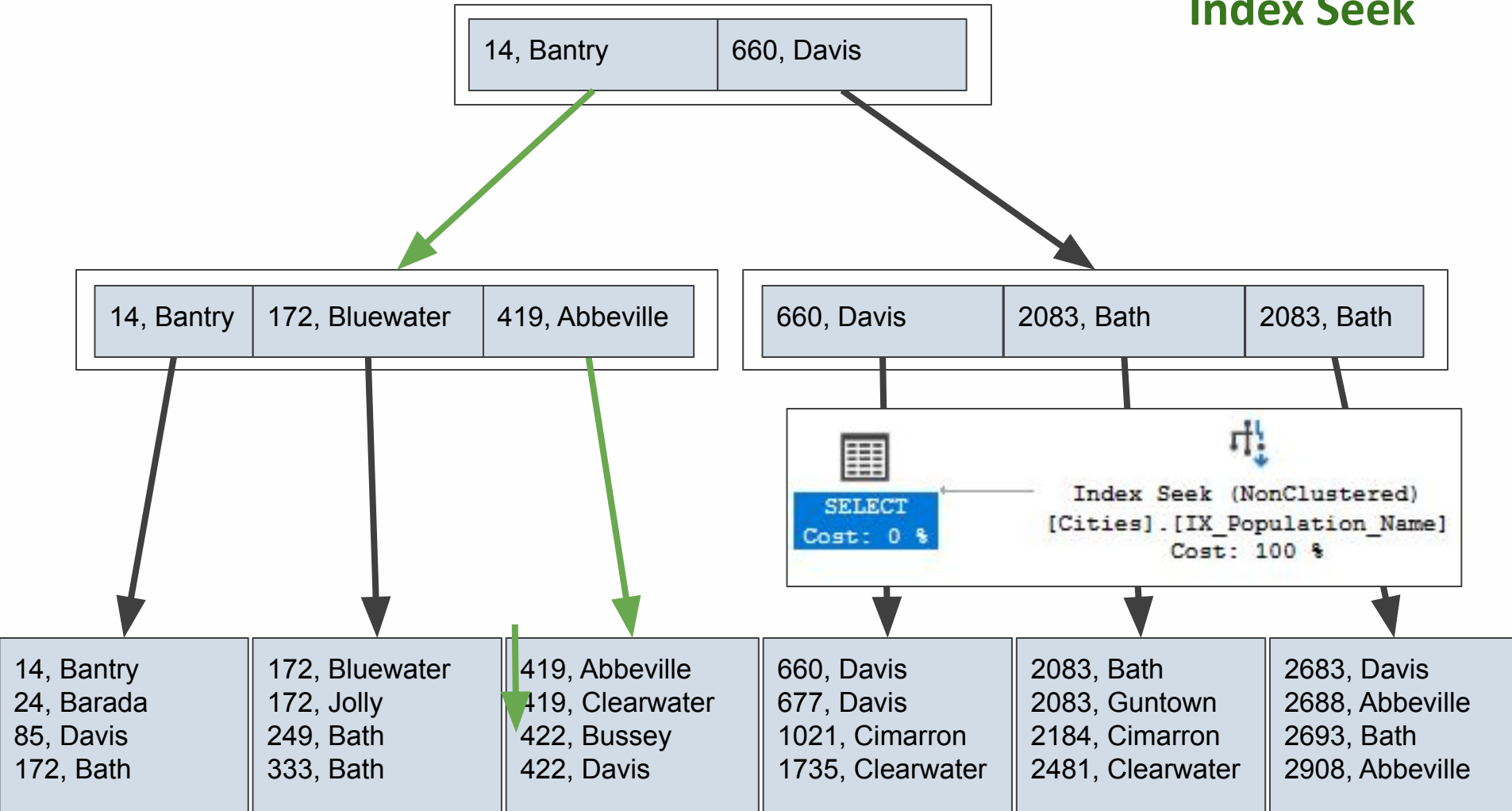
**WHERE CityName = 'Davis'****Index Scan**

**WHERE Population = 422 and CityName = 'Davis'**



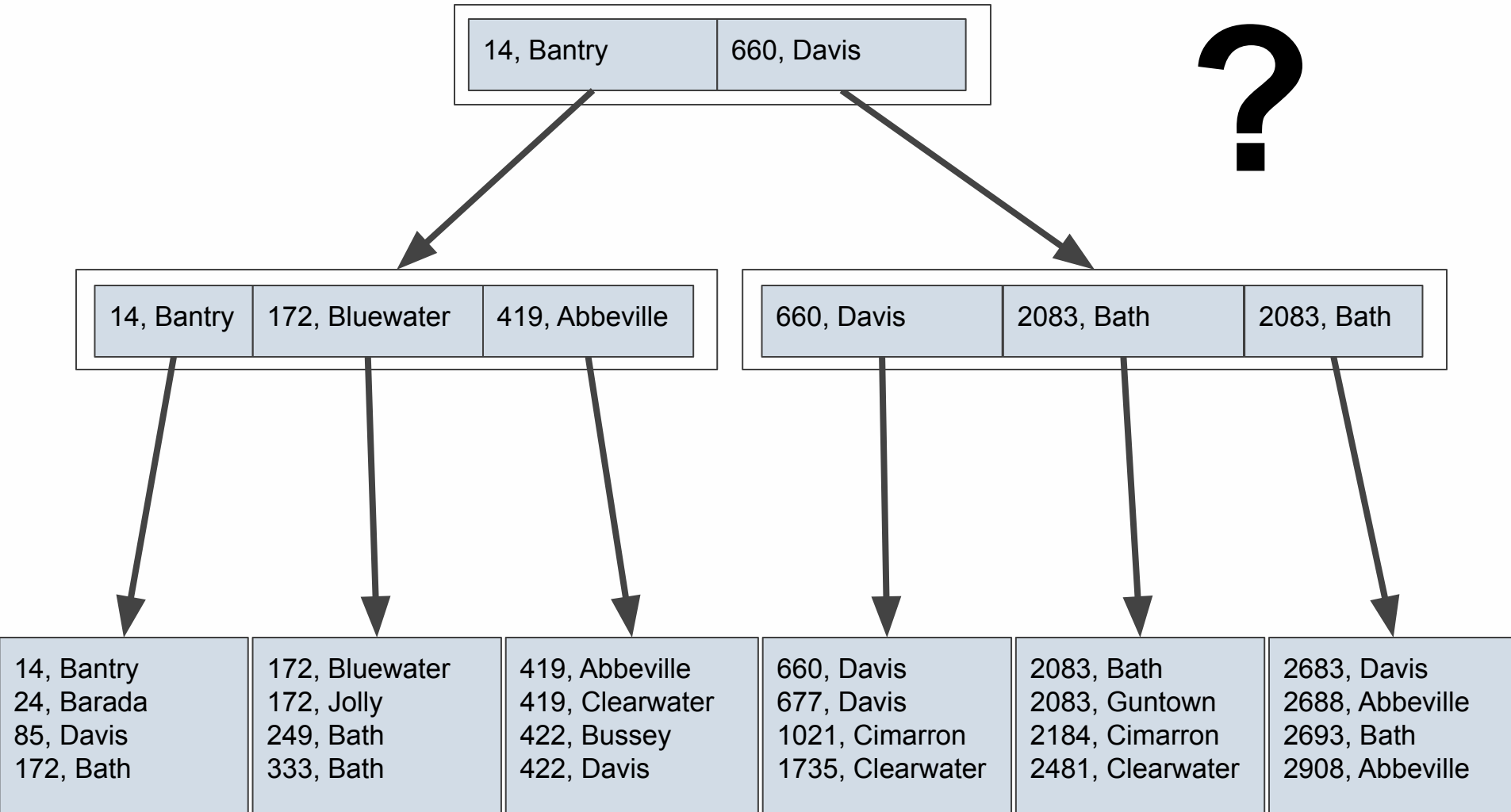
**WHERE Population = 422 and CityName = 'Davis'**

**Index Seek**



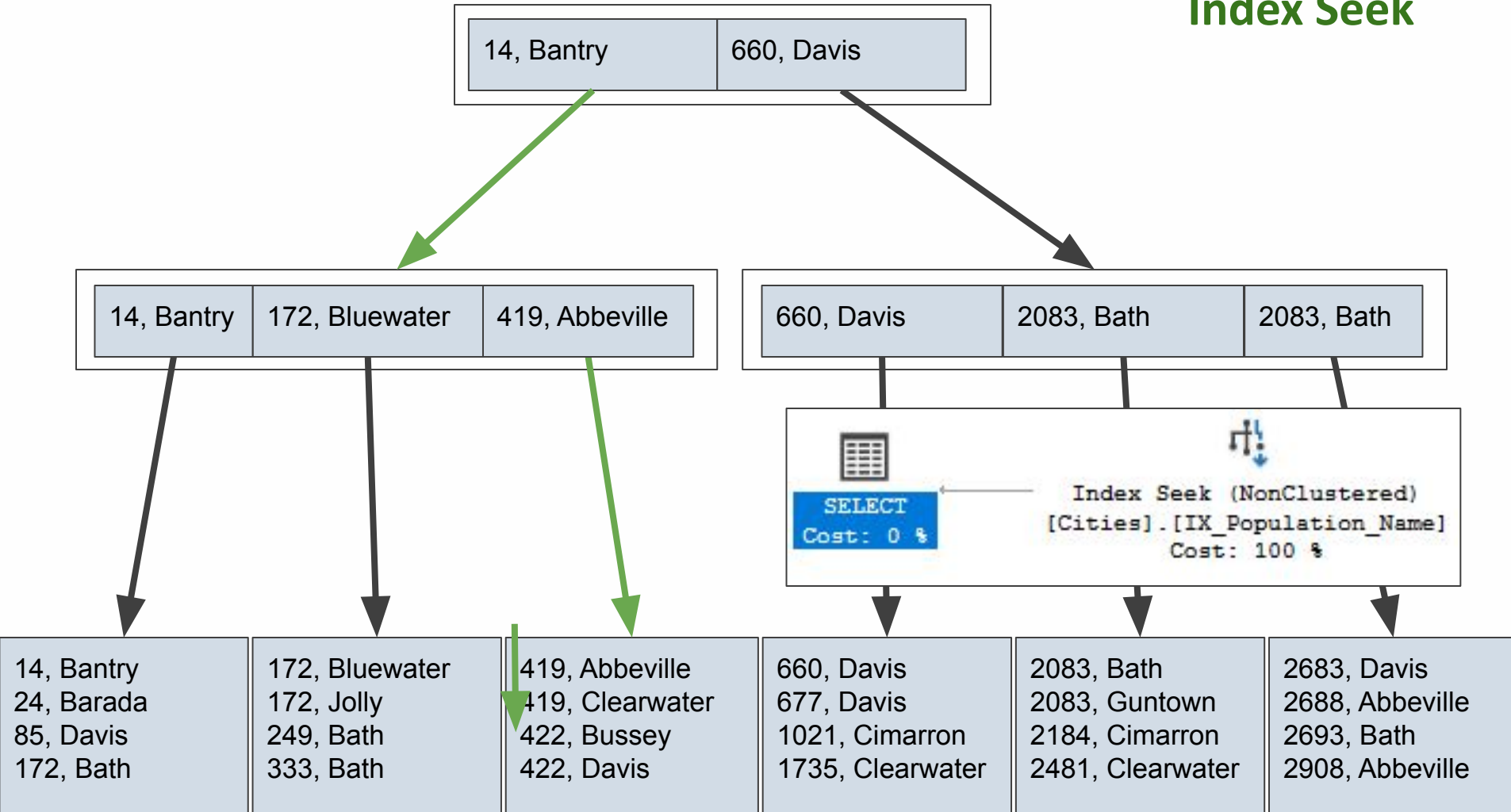


**WHERE Population > 422 and CityName = 'Davis'**



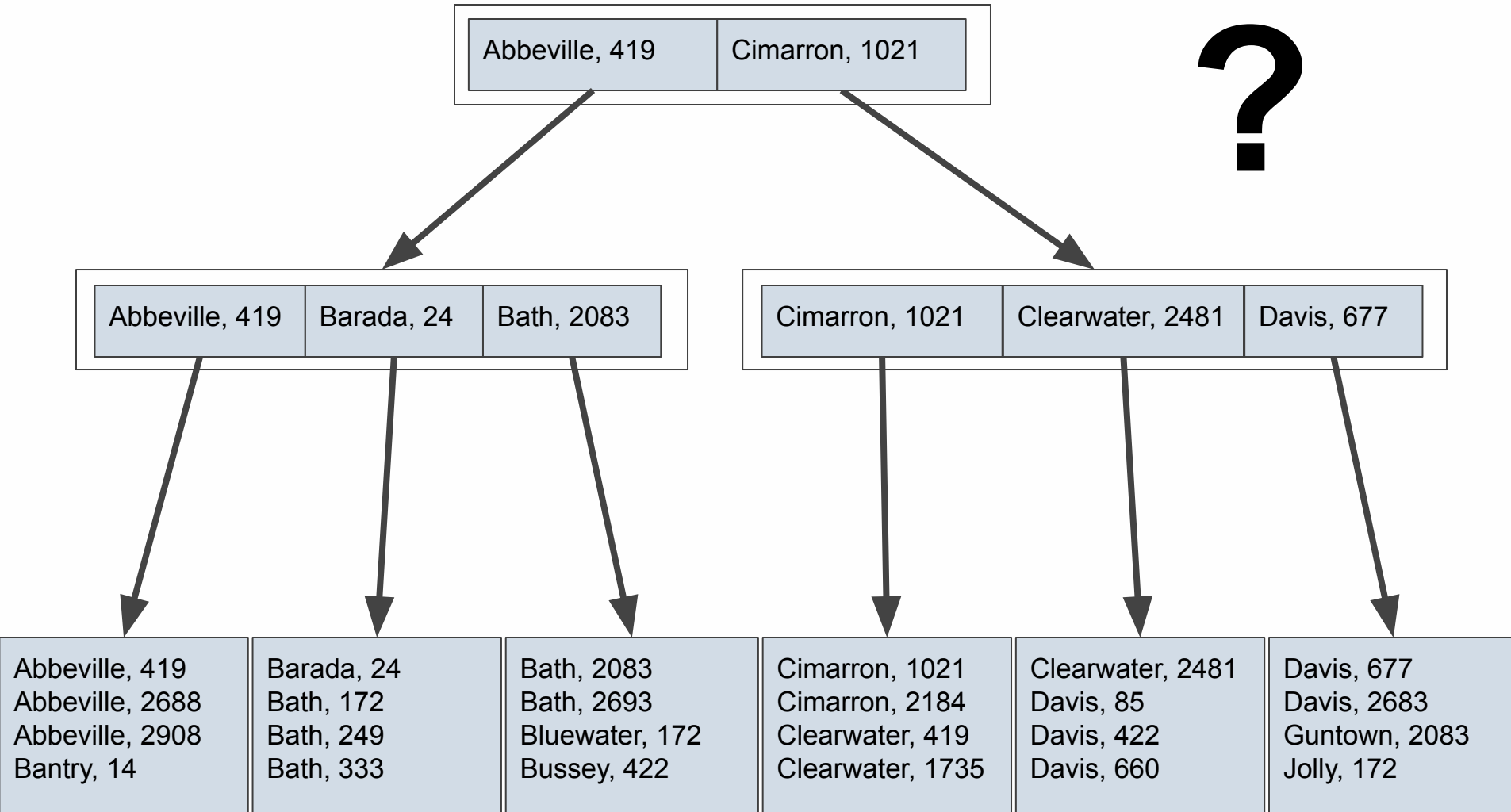
**WHERE Population > 422 and CityName = 'Davis'**

**Index Seek**



# А если другой индекс — IX\_Name\_Population

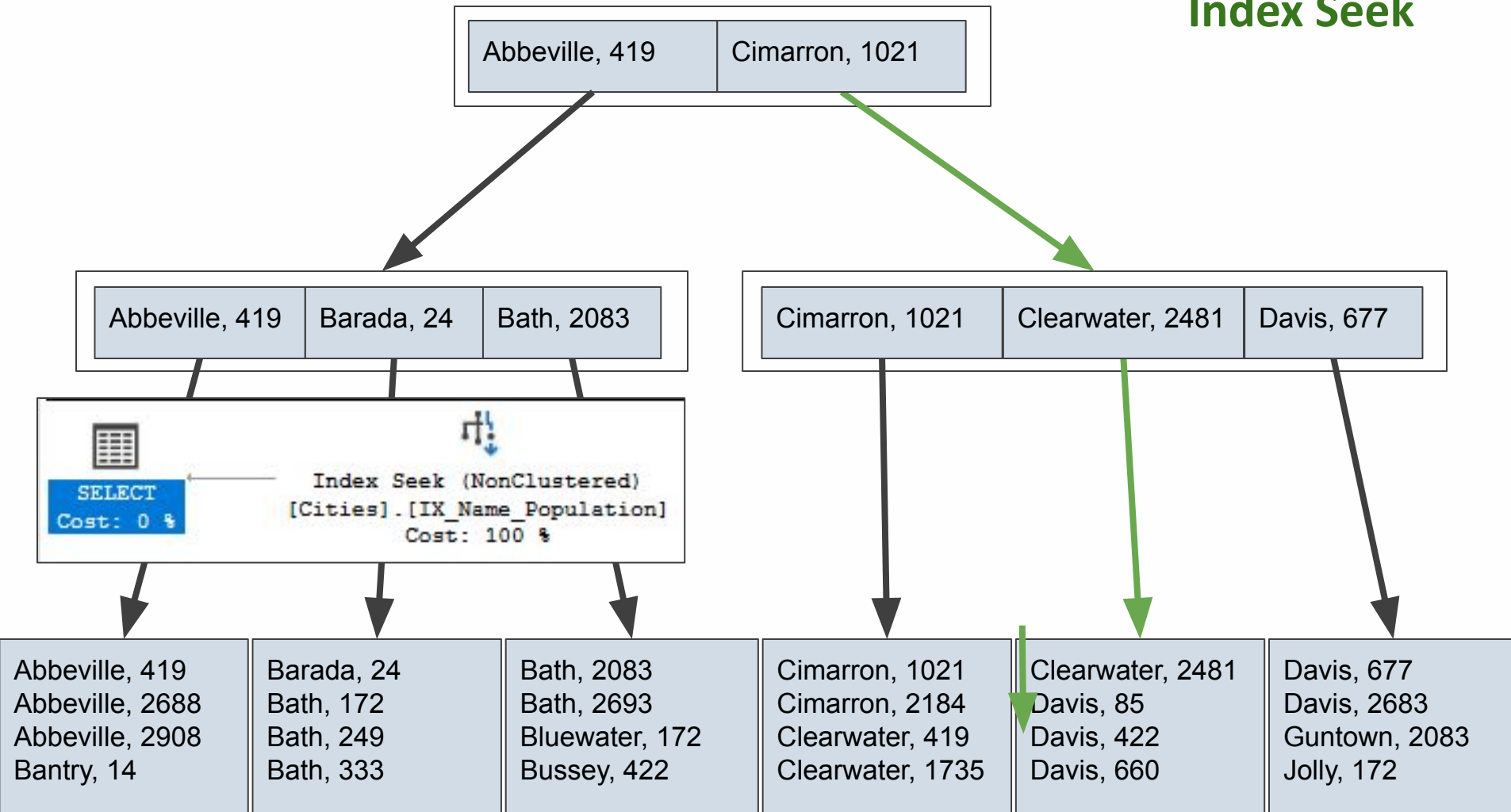
**WHERE Population = 422 and CityName = 'Davis'**



# А если другой индекс — IX\_Name\_Population

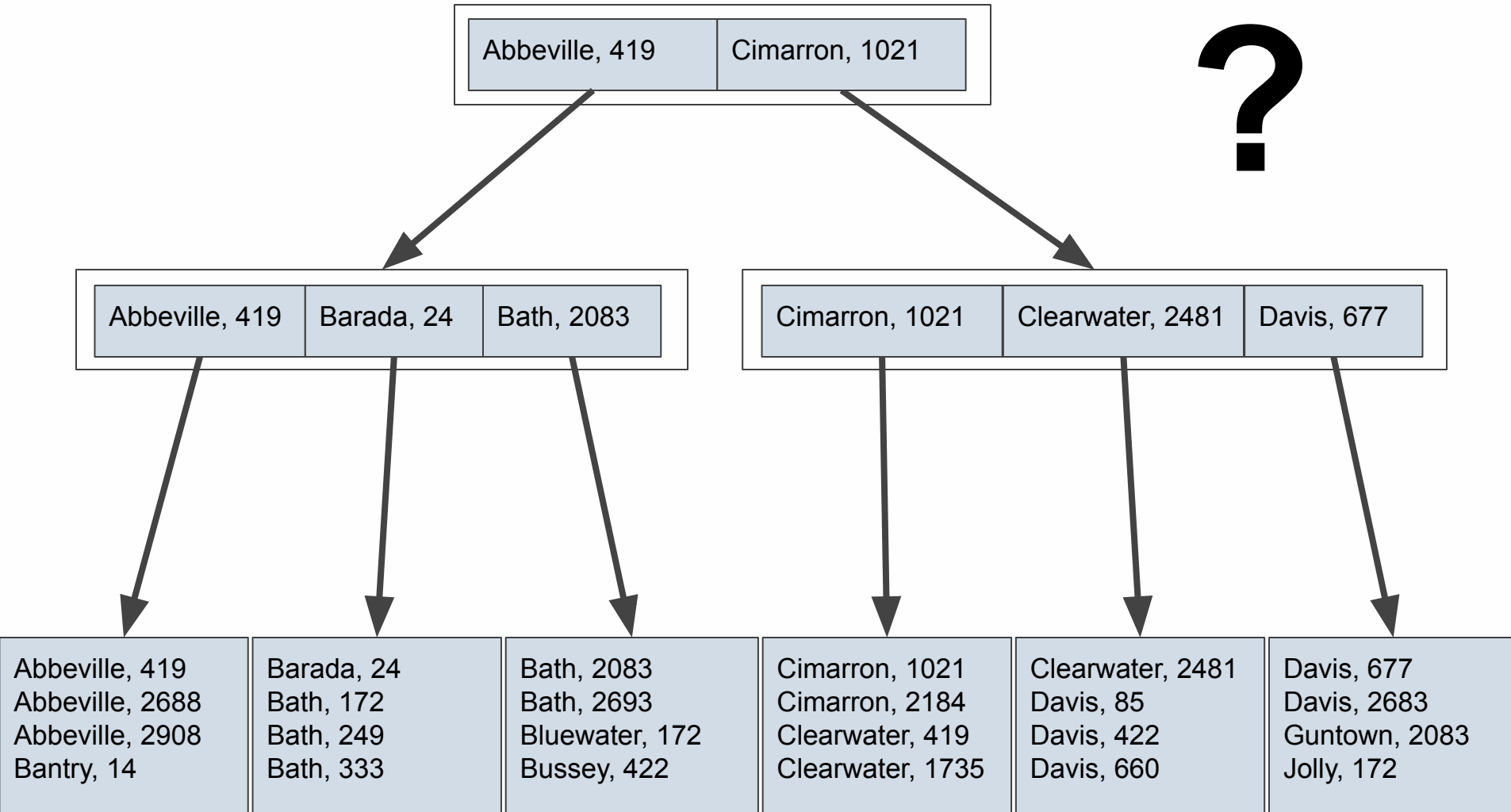
**WHERE Population = 422 and CityName = 'Davis'**

**Index Seek**



# А если другой индекс — IX\_Name\_Population

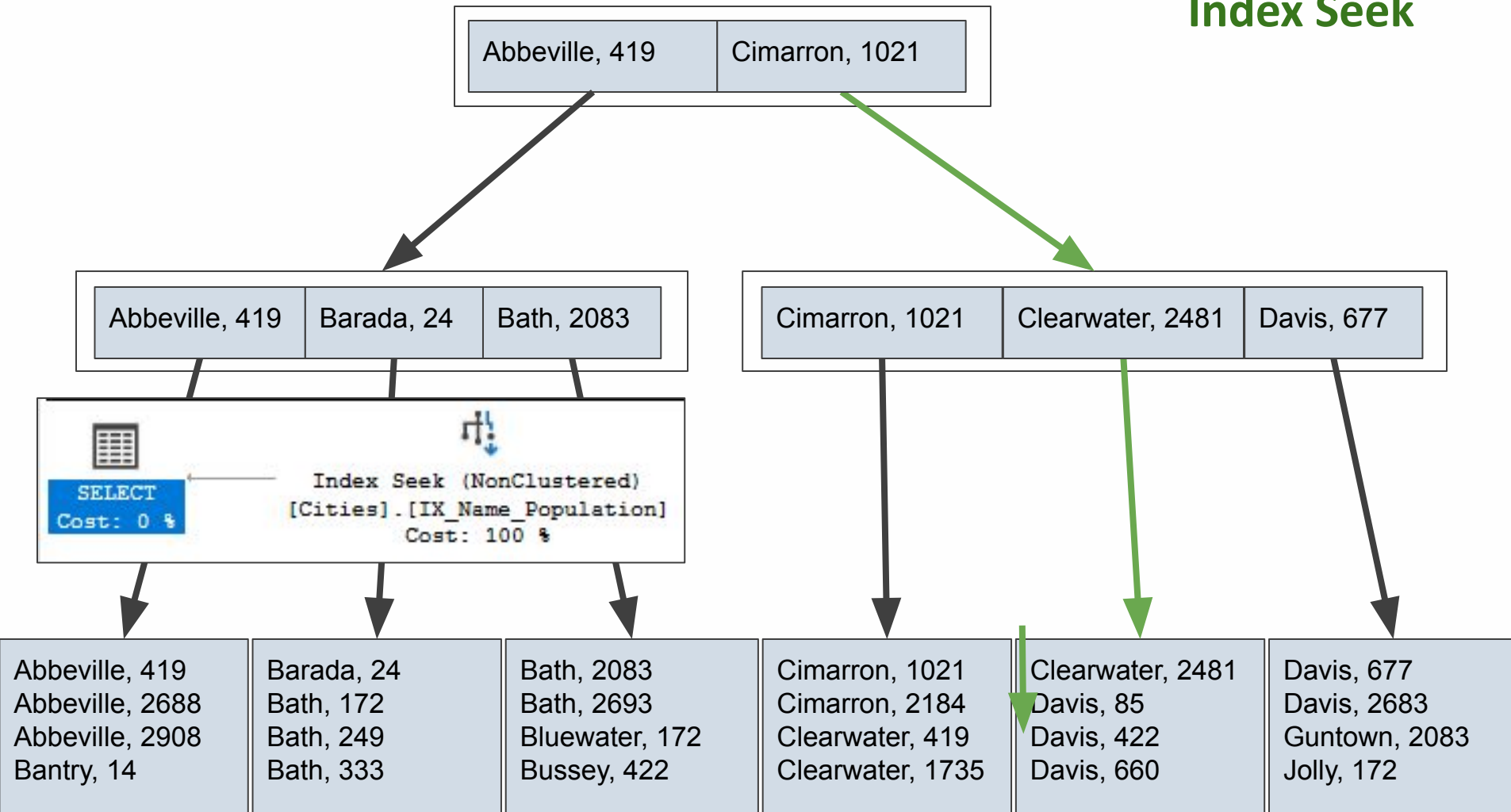
**WHERE Population > 422 and CityName = 'Davis'**



# А если другой индекс — IX\_Name\_Population

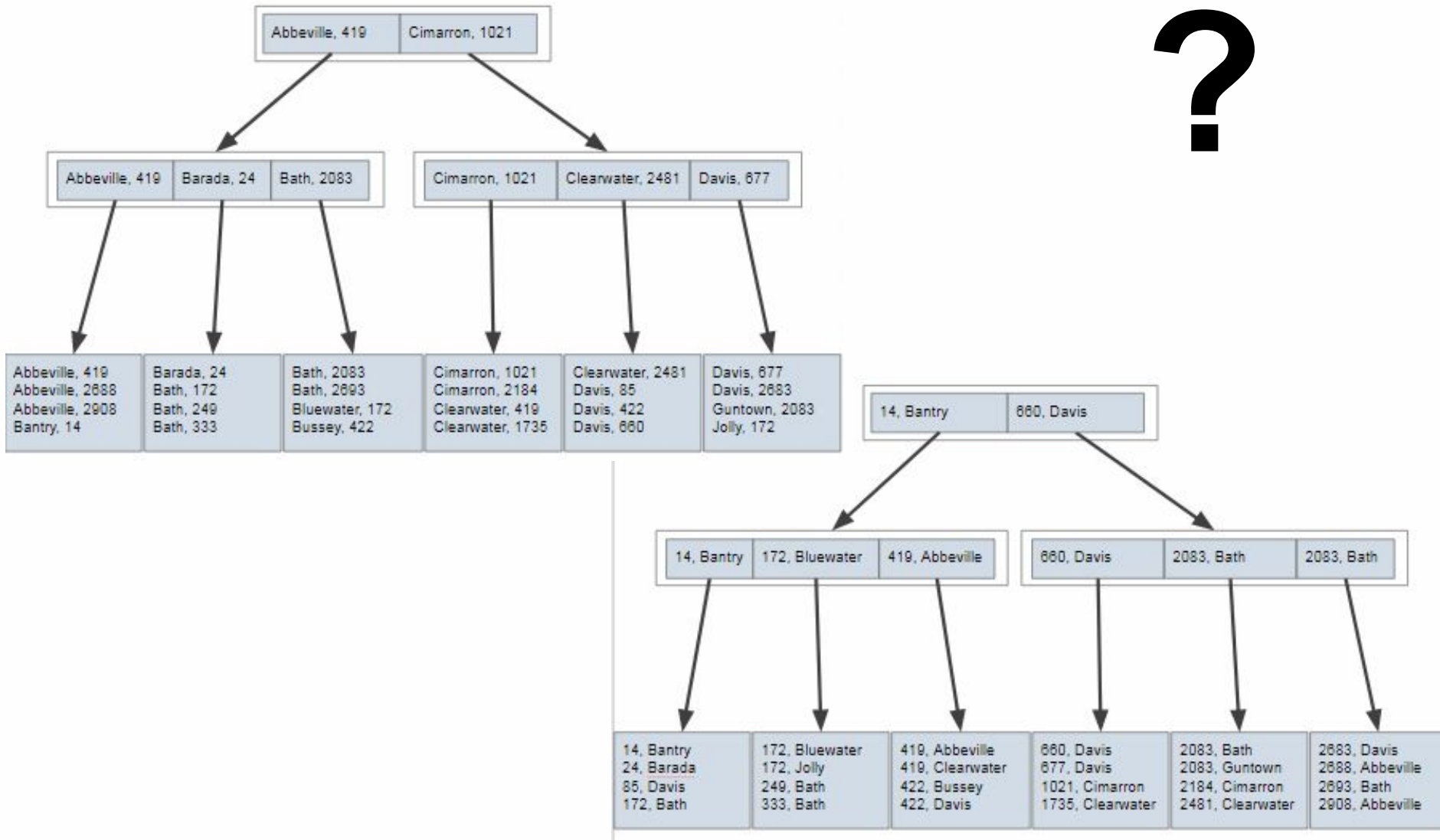
**WHERE Population > 422 and CityName = 'Davis'**

**Index Seek**

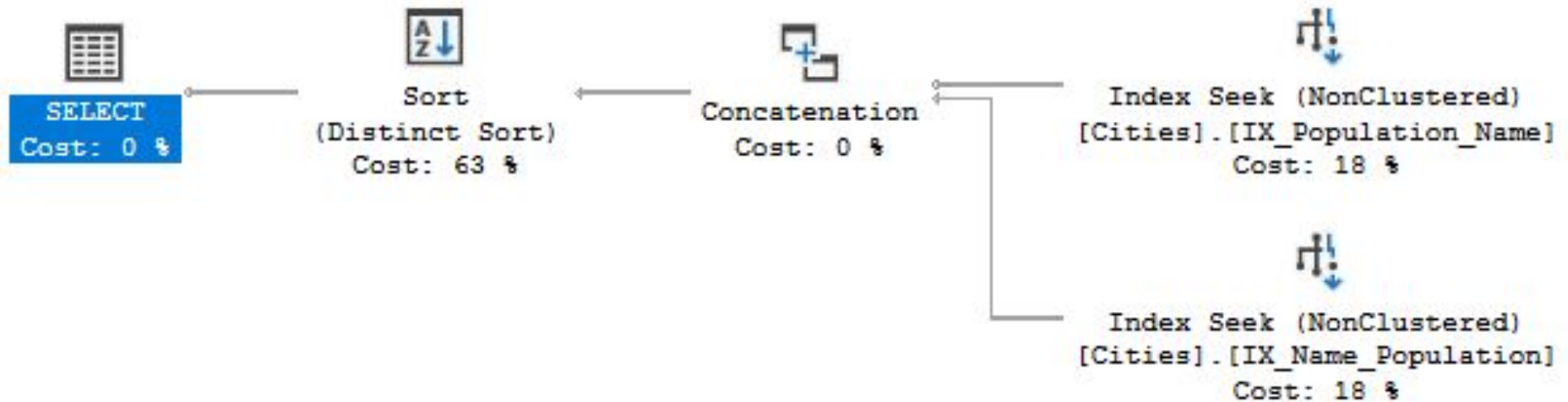


**WHERE Population = 422 or CityName = 'Davis'**

?



**WHERE Population = 422 or CityName = 'Davis'**

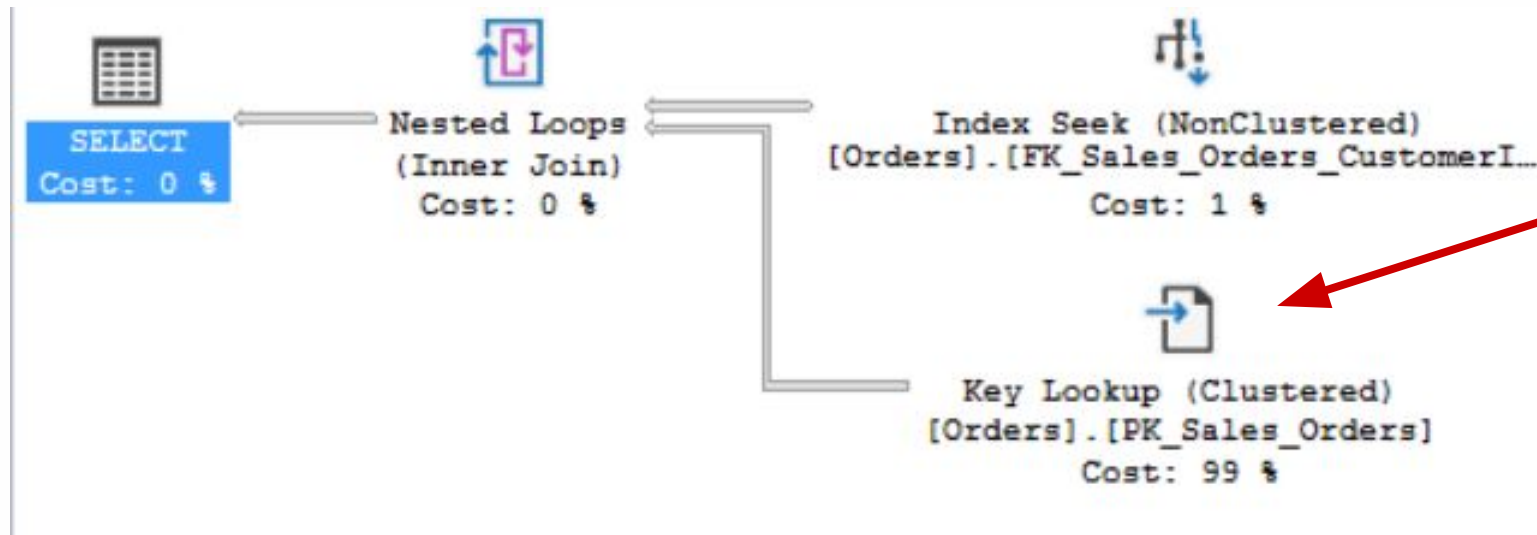




# 05.3

## Покрывающие индексы

- На листьях некластеризованного индекса также хранятся данные, кроме ссылки на данные
- Убираем Key Lookup, RID Lookup



# 05.4

## Фильтрованные индексы

- В некластерных индексах данные о всех строках. С помощью WHERE можно включить в индекс только нужные строки.

Например:

- WHERE Enabled = 1
- WHERE Column IS NOT NULL
- Уменьшение размера индекса.
- Улучшение производительности запроса
- Сложные ограничения UNIQUE

**Когда может использоваться  
индекс, а когда нет  
SARGable**

Вспомним типы JOIN.

В каких видах JOIN для соединения может использоваться индекс?

1. Nested loop (вложенные циклы)
2. Merge join (слияние)
3. Hash join (хэширование)



Вспомним типы JOIN.

В каких видах JOIN для соединения может использоваться индекс?

1. Nested loop (вложенные циклы)
2. Merge join (слияние)
3. Hash join (хэширование)

- Search ARGguments able
- Можно ли в условии WHERE использовать индексы или нет
- Надо стремиться создавать запросы, которые SARGable
- Почитать
  - [Можно ли, добавив индекс, сделать запрос SARGable?](#)
  - [How to Search and Destroy Non-SARGable Queries on Your Server](#)
  - [How to use sargable expressions in T-SQL queries; performance advantages and examples](#)



```
SELECT FirstName  
FROM Dummy_PersonTable  
where LEFT(FirstName,1)='K'
```



```
SELECT FirstName  
FROM Dummy_PersonTable  
where LEFT(FirstName,1)='K'
```

```
SELECT FirstName  
FROM Dummy_PersonTable  
where FirstName LIKE 'K%'
```

```
SELECT ModifiedDate  
FROM Dummy_PersonTable  
where YEAR(ModifiedDate)=2009
```



```
SELECT ModifiedDate  
FROM Dummy_PersonTable  
where YEAR(ModifiedDate)=2009
```

```
SELECT ModifiedDate  
FROM Dummy_PersonTable  
where ModifiedDate  
BETWEEN '20090101' AND '20091231'
```

# ДЕМО

## SARGable



- Не имеет смысла строить индексы для атрибутов, имеющих мало различных значений (низкая кардинальность)
- Составные индексы. Первым ставить с большой кардинальностью.
- Не имеет смысла создание индексов для таблиц малого размера.
- При массовой загрузке данных можно отключать индекс, после загрузки его включать / пересоздавать.
- Учитывать тип нагрузки (OLAP vs OLTP)
- Выносить в отдельные файлы (файловые группы)
- В запросе хотя бы одно условие в WHERE должно поддерживаться индексом
- Можно всегда сразу создавать на FK
- Про анализ использования индексов еще впереди

SQL Server при выборе метода доступа к данным (scan, seek) старается выбрать тот, когда **читается меньше данных**

Рекомендуется прочитать статью

<https://habr.com/ru/company/otus/blog/504144/>

"Почему SQL Server не гарантирует сортировку результатов без ORDER BY"

# 06

## Домашнее задание



Думаем какие запросы у вас будут в базе и добавляем для них индексы. Проверяем, что они используются в запросе.

В конце модуля "предзащита" проекта

- Диаграмма БД
- TSQL-код создания БД
- Показать основные/интересные/проблемные моменты

- Какие виды индексов вы запомнили?
- Какие вы бы применили у себя в проекте

**Пройдите, пожалуйста  
опрос**

**Спасибо  
за внимание!**

