In this part:

- natural key, (update, delete,)
- index,
- functions

Indexing

Indexing is the way to get an unordered table into an order that will maximize the query's efficiency while searching.

When a table is unindexed, the order of the rows will likely not be discernible by the query as optimized in any way, and your query will therefore have to search through the rows linearly. In other words, the queries will have to search through every row to find the rows matching the conditions. As you can imagine, this can take a long time. Looking through every single row is not very efficient.

For example, the table below represents a table in a fictional datasource, that is completely unordered.

company_id	unit	unit_cost
10	12	1.15
12	12	1.05
14	18	1.31
18	18	1.34
11	24	1.15
16	12	1.31
10	12	1.15
12	24	1.3
18	6	1.34
18	12	1.35
14	12	1.95
21	18	1.36
12	12	1.05
20	6	1.31
18	18	1.34
11	24	1.15
14	24	1.05

If we were to run the following query:

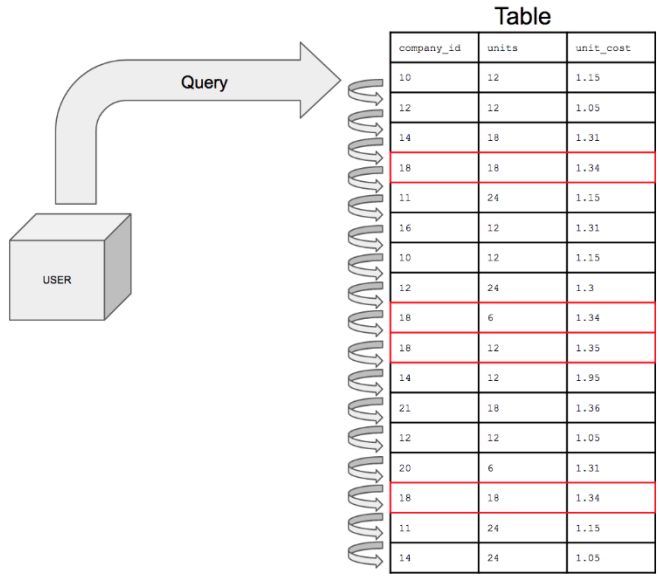
```
SELECT
      company_id,
      units,
      unit_cost
FROM
      index_test
WHERE
      company_id = 18
```

The database would have to search through all 17 rows in the order they appear in the table, from top to bottom, one at a time. So to search for all of the potential instances of the company_id number 18, the database must look through the entire table for all appearances of 18 in the company_id column.

This will only get more and more time consuming as the size of the table increases. As the sophistication of the data increases, what could eventually happen is that a table with one billion rows is joined with another table with one billion rows; the query now has to search through twice the amount of rows costing twice the amount of time.

You can see how this becomes problematic in our ever data saturated world. Tables increase in size and searching increases in execution time.

Querying an unindexed table, if presented visually, would look like this:



What indexing does is sets up the column you're search conditions are on in a sorted order to assist in optimizing query performance.

With an index on the company_id column, the table would, essentially, "look" like this:

company_id	unit	unit_cost
10	12	1.15
10	12	1.15
11	24	1.15
11	24	1.15
12	12	1.05
12	24	1.3
12	12	1.05
14	18	1.31
14	12	1.95
14	24	1.05
16	12	1.31
18	18	1.34
18	6	1.34
18	12	1.35
18	18	1.34
20	6	1.31
21	18	1.36

Now, the database can search for company_id number 18 and return all the requested columns for that row then move on to the next row. If the next row's comapny_id number is also 18 then it will return the all the columns requested in the query. If the next row's company_id is 20, the query knows to stop searching and the query will finish.

How does Indexing Work?

In reality the database table does not reorder itself every time the query conditions change in order to optimize the query performance: that would be unrealistic. In actuality, what happens is the index causes the database to create a data structure. The data structure type is very likely a B-Tree. While the advantages of the B-Tree are numerous, the main advantage for our purposes is that it is sortable. When the data structure is sorted in order it makes our search more efficient for the obvious reasons we pointed out above.

When the index creates a data structure on a specific column it is important to note that no other column is stored in the data structure. Our data structure for the table above will only contain the the company_id numbers. Units and unit_cost will not be held in the data structure.

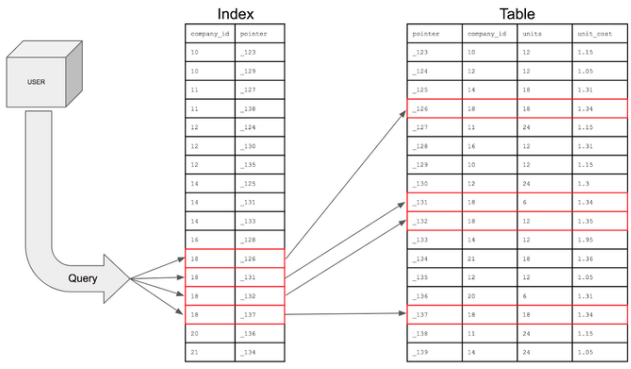
How Does the Database Know What Other Fields in the Table to Return?

Database indexes will also store pointers which are simply reference information for the location of the additional information in memory. Basically the index holds the company_id and that particular row's home address on the memory disk. The index will actually look like this:

company_id	unit	unit_cost
10	12	1.15
10	12	1.15
11	24	1.15
11	24	1.15
12	12	1.05
12	24	1.3
12	12	1.05
14	18	1.31
14	12	1.95
14	24	1.05
16	12	1.31
18	18	1.34
18	6	1.34
18	12	1.35
18	18	1.34
20	6	1.31
21	18	1.36

With that index, the query can search for only the rows in the company_id column that have 18 and then using the pointer can go into the table to find the specific row where that pointer lives. The query can then go into the table to retrieve the fields for the columns requested for the rows that meet the conditions.

If the search were presented visually, it would look like this:



Recap

- Indexing adds a data structure with columns for the search conditions and a pointer
- The pointer is the address on the memory disk of the row with the rest of the information
- The index data structure is sorted to optimize query efficiency
- The query looks for the specific row in the index; the index refers to the pointer which will find the rest of the information.
- The index reduces the number of rows the query has to search through from 17 to 4

Functions

Function	Opis
ABS	Returns the absolute value of a number
ACOS	Returns the arc cosine of a number
ASIN	Returns the arc sine of a number
ATAN	Returns the arc tangent of a number
ATN2 (ATAN2(y,x)	Returns the arc tangent of two numbers
AVG	Returns the average value of an expression
CEILING	Returns the smallest integer value that is >= a number
COUNT	Returns the number of records returned by a select query
COS	Returns the cosine of a number
COT	Returns the cotangent of a number
DEGREES	Converts a value in radians to degrees
EXP	Returns e raised to the power of a specified number/ e^x
FLOOR	Returns the largest integer value that is <= to a number
LOG	Returns the natural logarithm of a number, or the logarithm of a number to a specified base
LOG10	Returns the natural logarithm of a number to base 10
MAX	Returns the maximum value in a set of values
MIN	Returns the minimum value in a set of values
PI	Returns the value of PI / π
POWER	Returns the value of a number raised to the power of another number
RADIANS	Converts a degree value into radians
RAND	Returns a random number
ROUND	Rounds a number to a specified number of decimal places
SIGN	Returns the sign of a number
SIN	Returns the sine of a number
SQRT	Returns the square of a number

SQUARE	Returns the square of a number
SUM	Calculates the sum of a set of values
TAN	Returns the tangent of a number

SQL Server Funkcje daty

Function	Description
CURRENT_TIMESTAMP	Returns the current date and time
DATEADD	Adds a time/date interval to a date and then returns the date
DATEDIFF	Returns the difference between two dates
DATEFROMPARTS	Returns a date from the specified parts (year, month, and day values)
DATENAME	Returns a specified part of a date (as string)
DATEPART	Returns a specified part of a date (as integer)
DAY	Returns the day of the month for a specified date
GETDATE	Returns the current database system date and time
GETUTCDATE	Returns the current database system UTC date and time
ISDATE	Checks an expression and returns 1 if it is a valid date, otherwise 0
MONTH	Returns the month part for a specified date (a number from 1 to 12)
SYSDATETIME	Returns the date and time of the SQL Server
YEAR	Returns the year part for a specified date

SQL Server Advanced Functions

Function	Description
CAST	Converts a value (of any type) into a specified datatype
COALESCE	Returns the first non-null value in a list
CONVERT	Converts a value (of any type) into a specified datatype
CURRENT_USER	Returns the name of the current user in the SQL Server database
ISNULL	Return a specified value if the expression is NULL, otherwise return the expression
ISNUMERIC	Tests whether an expression is numeric
NULLIF	Returns NULL if two expressions are equal
SESSION_USER	Returns the name of the current user in the SQL Server database
SESSIONPROPERTY	Returns the session settings for a specified option
SYSTEM_USER	Returns the login name for the current user
USER_NAME	Returns the database user name based on the specified id

Funkcje tekstowe

Function	Description
ASCII	Return the ASCII code value of a character
CHAR	Convert an ASCII value to a character
CHARINDEX	Search for a substring inside a string starting from a specified location and return the position of the substring.
CONCAT	Join two or more strings into one string
CONCAT_WS	Concatenate multiple strings with a separator into a single string
DIFFERENCE	Compare the SOUNDEX() values of two strings
FORMAT	Return a value formatted with the specified format and optional culture
LEFT	Extract a given a number of characters from a character string starting from the left
LEN	Return a number of characters of a character string
LOWER	Convert a string to lowercase
LTRIM	Return a new string from a specified string after removing all leading blanks
NCHAR	Return the Unicode character with the specified integer code, as defined by the Unicode standard
PATINDEX	Returns the starting position of the first occurrence of a pattern in a string.
QUOTENAME	Returns a Unicode string with the delimiters added to make the input string a valid delimited identifier
REPLACE	Replace all occurrences of a substring, within a string, with another substring
REPLICATE	Return a string repeated a specified number of times
REVERSE	Return the reverse order of a character string
RIGHT	Extract a given a number of characters from a character string starting from the right
RTRIM	Return a new string from a specified string after removing all trailing blanks
SOUNDEX	Return a four-character (SOUNDEX) code of a string based on how it is spoken
SPACE	Returns a string of repeated spaces.
STR	Returns character data converted from numeric data.
STRING_AGG	Concatenate rows of strings with a specified separator into a new string
STRING_ESCAPE	Escapes special characters in a string and returns a new string with escaped characters
STRING_SPLIT	A table-valued function that splits a string into rows of substrings based on a specified separator.
STUFF	Delete a part of a string and then insert another substring into the string starting at a specified position.
SUBSTRING	Extract a substring within a string starting from a specified location with a specified length
TRANSLATE	Replace several single-characters, one-to-one translation in one operation.
TRIM	Return a new string from a specified string after removing all leading and trailing blanks

UNICODE	Returns the integer value, as defined by the Unicode standard, of a character.
UPPER	Convert a string to uppercase