|  |  |
| --- | --- |
| -  Sindy Saintclair  Tuesday, January 18, 2022  Lesson 5 – Indexes and Project | |
| **Learning Objectives and Questions** | **Notes and Answers** |
| **INDEXES** | Indexes deal with the ordering of your data and have a significant impact on the performance of your database. They are used to retrieve the data from the database very quickly and improves the read time of the data. On the other hand, it lengthens the write-time when creating a new record. Let’s explore why.  There are two particular types of indexes in SQL server to concern yourself with:   * **Clustered index**: this is the actual, physical order of the data as it is stored on disk. When you have a clustered index, the SQL server stores it by that column. You can only have one of these on table, and because of this, it is common that the primary key is also the clustered index. There are times you will want this not to be the case ,but the vast majority of situations this will be your default. Think of it like a dictionary: the data is ordered in alphabetical order throughout the entire book, and when a new word is added, it is not added at the end, but where it belongs alphabetically. * **Non-clustered index** creates an entirely different object within the table. It contains the column or columns selected for indexing and points to the table’s rows containing the data. The purpose of an index is to provide a way to expediently get your query results, without having to examine every row in a given table. The tradeoff of indexes is that when you create one, you copy all of the data you are indexing into a new structure with the order you have specified. Because of this ,if there are too many indexes on a table, you may very well have far more storage dedicated to indexes than the actual storage of the table itself. It’s not uncommon to see a table with many columns and 500mb of storage to have index space usage of over 1GB. Another drawback of too many indexes is that you may incur performance issues on heavily inserted/updated/deleted tables. That’s because every time you do one of these operations-the same operation must be completed on every index to ensure data integrity. So index intelligently, and as infrequently as possible! A Non-clustered index is like an index at the back of book: keywords and references are stored there with the page numbers of the material for quick reference. Anytime the book is updated, the index needs to be updated as well.   *Create Index*  The CREATE INDEX statement is used to create indexes in a table.  The below query will create an index on a table. In this case, duplicate values are allowed:  CREATE INDEX index\_name  ON table\_name (column1, column2, ...);  CREATE UNIQUE INDEX  This will create a unique index on a table. Duplicate values are not allowed:  CREATE UNIQUE INDEX index\_name  ON table\_name (column1, column2, ...);  Then, when you run a query on with the same specific columns you defined in the CREATE INDEX, it should pull in that data quicker than just running a query without an index. Make a note that your database isn’t extremely large, so you may not see a considerable change in the speed of the data being pulled in. When you start working with a database with millions of rows, that’s when indexes are vital and will help speed up the query.  Go ahead and run the following query:  CREATE INDEX filmListing  ON sakila.film (film\_id)  You have created an index when selecting only ‘film\_id’ in a query from the ‘film’ table.  *Explain Query Plan*  The EXPLAIN QUERY PLAN will show you what is happening when a query is run.  If you run:  SELECT film\_id FROM sakila.film;  You will get a list of all film id’s from the film table (1000 rows). Want to find more? You can click on the Execution Plan tab to see more!  Arrow pointing towards execution plan tab on the right hand side.  You will see that it is showing what index is being used. This is a way you can check to see if your index is being used correctly or at all, and it’s also a way to get information about how long your query takes.  Full Index Scan of 112 seconds on the film_id index leading to query_block #1. |
| **Common SQL Functions** | SQL has many built-in functions that help with processing string or numeric data.  Below are some of the typical SQL functions:  *Min and Max*  The min( ) function in SQL will return the smallest value of the selected column. If you want to find the smallest amount paid within the payment table, you can run the following query:  SELECT min(amount) FROM sakila.payment  And that would give the following output:  Min total. Table with a column heading min amount. The first cell under the heading has the value zero point zero zero.  *Figure 5-2: Min Total*  But if you need to see the most spent within the invoices column, replace ‘min’ with ‘max’ like below:  SELECT max(amount) FROM sakila.payment  Which would give the output:  Max total. Window with a line of code that reads select max total from invoices. There is an execute button in the lower right corner of the window.  Figure 5-3: Max Total  *Sum*  The sum( ) function will return the total sum of a numeric column. Continuing with the payment table, you can find the sum of all totals paid within the payments by running the following query:  SELECT sum(amount) FROM sakila.payment  And the output will be:  Sum total. Table with a column heading sum amount. The first cell under the heading has the value 67 thousand 416 point 51.  Figure 5-4: Sum Total  *Avg*  The avg( ) function will return the average value of a numeric column. Instead of finding the sum of the Total column, find the average:  SELECT avg(amount) FROM sakila.payment  Output:  Average total. Min total. Table with a column heading avg amount. The first cell under the heading has the value four point two zero zero six six seven.  Figure 5-5: Average Total  *Count*  The count( ) function will return the number of rows that match certain criteria. Run the following query:  SELECT count(amount) FROM sakila.payment  WHERE amount < 4.00  And the output will be:  Count total. Min total. Table with a column heading count amount. The first cell under the heading has the value 8 3 0 3.  Figure 5-6: Count Total |
| **Group By** | THE SQL GROUP BY statement is often used with the functions mentioned earlier in this lesson (MIN, MAX, SUM, AVG, COUNT). The results of these functions can then be grouped using one or more columns. Below is the syntax:  SELECT column\_name(s) FROM table\_name  WHERE condition  GROUP BY column\_name(s)  Using GROUP BY, run the following query to list the number of addresses in each district:  SELECT count(address\_id), district FROM sakila.address  GROUP BY district  ORDER BY count(address\_id) DESC  Breaking down the above query, you can see that you are selecting a count of the address IDs column and the district column from the table ‘address’. Then you are grouping everything by the column ‘district’. Then, the output is being ordered in descending order by the count of the address IDs. The output will be:  Group by. Table with the headings count address and district. There are several values entered under each heading.  Figure 5-7: Group By  GROUP BY can be a bit complicate to understand. Below are a few more examples of GROUP BY:  *Example 1*  SELECT sakila.customer.customer\_id,sakila.customer.first\_name, sakila.customer.last\_name, COUNT(rental\_id) FROM sakila.rental  INNER JOIN sakila.customer USING (customer\_id)  GROUP BY sakila.customer.customer\_id  Above, we are selecting four columns: customer\_id, first\_name, and last\_name from the customer table, and the rental\_id from the rental table, which is being counted. We are selecting from the rental table and joining the customer table using the customer\_id column, which exists in both tables. We are then grouping by the customer\_id. Below is the output:  Group by Join. Table with the headings customer IN, first name, last name, count (rental ID.). There are several entries under each heading.  Figure 5-8: Group By Join  *Example 2*  SELECT customer\_id, sum(amount) AS TotalRentalAmount FROM sakila.payment  GROUP BY customer\_id  Above, we are adding the amounts together from the payment table and then grouping it all by the customer\_id. We are also using the AS keyword to rename the count column to reflect the total amount spent by each customer for their rentals. The output shows:  Group by sum. Table with the headings customer ID and total rental amount. There are several entries under each heading.  Figure 5-9: Group By Sum  *Example 3*  SELECT customer\_id, sum(amount) AS Total, count(rental\_id) AS NoOfRentals FROM sakila.payment  GROUP BY customer\_id  HAVING count(rental\_id) > 40  Above, we include a new keyword HAVING. This word was included in SQL because the WHERE statement does not work when using an aggregate function (i.e., sum, count, max, min, etc.). HAVING works the same way as WHERE because it defines a condition. The only difference is the condition is an aggregate function.  The query being run above shows the customer\_id of each customer who has rented over 40 rentals and the total amount they have spent on rentals.  Group by Having. Table with the headings customer ID, total, and number of rentals. There are several entries under each heading.  Figure 5-10: Group By Having |
| **Maintaining Database Security** | In order to maintain a database’s security, there are some things of which you will need to be aware. You may not do these things yourself, depending on your role, but you should be aware of what is required to maintain database security regardless.   * Ensure that everyone who has access to the database requires access * Ensure default passwords are changed * Remove public access * Ensure that user groups/roles are assigned correctly. No one should be able to do more with the database than they need to. * Divide up database administrative duties, so everything does not reside on just one person * Document your database structure and configurations * Create a testing environment, so that when you are making changes, you can double check the work before going live * Regularly do process checks on your data to make sure that everything is high quality and no breaches have occurred * Have a recovery plan for when disaster strikes * Know what do when a data breach has occurred. |