ביצועים - SQL

מהם ביצועים?

במסגרת הסיכום ניתן טעימה, ולא כיסוי מלא למה הם "ביצועים".

במסדי נתונים, "ביצועים" מתייחסים למהירות וליעילות שבה המערכת יכולה להריץ שאילתות ולטפל בנתונים.

ביצועים טובים פירושם קבלת התוצאות הנדרשות במהירות וללא בזבוז משאבים.

ביצועים גרועים פירושם שהשאילתות לוקחות יותר מדי זמן, צורכות יותר מדי זיכרון או CPU, או מאטות את המערכת כולה.

הביצועים תלויים בגורמים כמו אופן כתיבת שאילתות, גודל הטבלאות, האם קיימים אינדקסים, ומהירות החומרה.

האם לשפר ביצועים?

דונלד קנות' לימד אותנו כי "אופטימיזציה מוקדמת היא שורש כל רע".

הדבר נובע משתי סיבות. ראשית, פעמים רבות אנחנו לא יודעים איפה נצטרך לשפר ביצועים. תתכן שאילתא מהירה יחסית המבוצעת פעמים רבות או בקטעים קריטים שאותה נצטרך לשפר. לעומתה, עושיה להיות שאילתה ארוכה יותר שלא יהיה אכפת לנו שתרוץ בלילה.

בעיה חמורה יותר היא השפעת השיפור. אם יש מימוש שהוא מהיר, פשוט ואלגנטי יותר ממימוש כואב קודם, ההחלטה לשפר קלה. בעיה היא שלרוב יש לנו פתרון מהיר יותר שהוא מסובך יותר, נוטה יותר לבאגים או קשה לתחזוקה. אם הגעה לרמת ביצועים מסויימת היא הכרח, אנו נפגע באיכות המערכת כדי לעמוד באילוץ הביצועים.

האם לשפר ב"רמאות"?

אנו נלמד איך ניתן לשפר ביצועים של שאילתא. לפני שנגיע לאלו, כדאי לבדוק האם ניתן לרמות, כלומר לשפר בלי לשפר את השאילתא.

דוגמאות לכך הן:

- החלטה לא לבצע שיפור. אם הביצועים מספקים, ההחלטה תחסוך זמן ותהפוך אתכם לאנשים עם ביצועים גבוהים יותר. אי ביצוע הוא הביצוע הכי מהיר.
- חומרה חזקה יותר
- דגימה וסינון (חישוב על אלף מקרים ולא מיליון)
- חישוב מקדים (Pre-compute) (חישוב בלילה ולא בהצגת עמוד למשתמש). זיכרו שזמן נמדד ביחידות סמנטיות, לא בשניות.

ככלל אצבע, זמן אנושי הרבה יותר יקר מזמן מחשב. רמאות לרוב לוקחת פחות זמן ויש רמת ביטחון גבוהה יותר בהצלחה. אם ה"רמאות" מתאימה, היא לרוב הפתרון העדיף.

אילו גורמים משפיעים על ביצועי השאילתות?

1. יעילות השאילתא - כמה השאילתא שנכתבה יעילה (האם יש יותר Joinים מהדרוש? האם יש דרך יותר חכמה לשלוף את המידע שצריך?)
2. גודל הטבלאות - באופן טבעי, טבלאות גדולות יותר דורשות יותר זמן לעיבוד או לחיפוש ולכן יכולות להשפיע על ביצועי שאילתא.
3. חומרה - מהירות ה-Cpu, כמות הזיכרון וסוג הדיסק יכולים לגרום לשאילתות לרוץ מהר יותר (או לאט יותר, אם לא טובים).
4. נרמול - עודף נרמול עלול לגרום לפעולות שידרשו קבועי זמן ארוכים יותר להרצת השאילתות.
5. שימוש באינדקסים (מפורט בהמשך)

איך ניתן לזהות אילו גורמים משפיעים על הביצועים?

אפשר להריץ ולמדוד, להסתכל על התוכנית ולעיתים אף להשתמש בכלים ייעודיים אחרים. לדוגמה - ב-MySQL קיים מנגנון מובנה בשם **Performance Schema** שמנטר בצורה מפורטת את פעילות מסד הנתונים.

מהן פונקציות Explain ו- Explain Analyze?

EXPLAIN היא פקודה שמראה את תוכנית הביצוע שמנוע ה־SQL מתכנן לשאילתה (אילו טבלאות ייסרקו, באיזה סדר, איזה סוג הצטרפות (JOIN), ואם ישתמש באינדקסים.)

EXPLAIN ANALYZE גם מריץ את השאילתה בפועל וגם מציג את הנתונים האמיתיים: כמה זמן לקח לכל שלב וכמה שורות באמת עובדו.

ההבדל: EXPLAIN הוא תחזית בלבד ואינו מריץ את השאילתה, בעוד EXPLAIN ANALYZE מריץ אותה בפועל ומחזיר מדידות אמת.

השוואה בין הערכות של EXPLAIN לבין התוצאות האמיתיות של EXPLAIN ANALYZE עוזרת לגלות צווארי בקבוק ולשפר ביצועים.

דוגמא לתוצאה שליפת Explains

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	movies	NULL	ALL	NULL	NULL	NULL	NULL	385202	100.00	NULL

דוגמא לשליפת Explain Analyze

	EXPLAIN
▶	-> Table scan on movies (cost=39771 rows=385202) (actual time=9.69..296 rows=388269 loops=1)

מהו אינדקס וכיצד משפיע על ביצועי השאילתות?

אינדקס הוא כמו תוכן עניינים בספר: הוא שומר מבנה מסודר שמאפשר למערכת לדעת איפה בדיוק הנתון נמצא בלי לעבור על כל הטבלה.

למה זה חשוב?

בלי אינדקס - חיפוש חייב לעבור על כל השורות (Full Table Scan) וזה איטי.

עם אינדקס - המערכת קופצת מהר יותר למקום הרלוונטי, והחיפוש הרבה יותר מהיר.

איך זה משפיע על זמן?

- בלי אינדקס: זמן החיפוש גדל לינארית עם כמות הנתונים (O(n)).

- עם אינדקס: זמן החיפוש לוגריתמי או אפילו קבוע (O(log n) / (1)).(העץ הוא הלוגירתמי)

כלומר גם אם יש מיליוני רשומות, עדיין נוכל למצוא תוצאה מהר מאוד.

חסרונות:

- תופס עוד מקום בזיכרון/דיסק.
- מאט פעולות כתיבה (INSERT / UPDATE / DELETE) כי צריך לעדכן גם את האינדקס. ביצועי האינדקס מורעים עם שינויים. לעיתים עדיף למחוק וליצור מחדש את האינדקס.

איך זה עובד בפועל?

נניח שיש לנו טבלת **movies** עם 388269, וטבלת **Actors** עם השחקנים בכל סרט כוללת 51504, רשומות.

אם נבצע שליפה רגילה (שבה קיימים אינדקסים מראש)

```
SELECT *
FROM actors
WHERE id = 12345;
```

איך המערכת מתכננת לבצע את השליפה הזו? נשתמש ב-Explain

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	actors	NULL	const	PRIMARY	PRIMARY	4	const	1	100.00	NULL

ניתן לראות שהמערכת משתמשת ב-PK כדי לשלוף את השורה הרלוונטית. אפשר לראות שקיימת רק שורה אחת כזאת.

מה אם נבצע פעולה חישובית על ה-Actor_id? לדוגמה

```
select * from actors
where md5(id) = md5(12345);
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	actors	NULL	ALL	NULL	NULL	NULL	NULL	816387	100.00	Using where

כאן ניתן לראות שהמערכת לא משתמשת ב-PK.

זמני השליפה הראשונה לעומת השליפה השנייה:

0.000 sec / 0.000 sec

0.907 sec / 0.000 sec

(נציין כי זמני הריצה הם לא באמת 0, אלא מעוגלים)

נראה את ההבדלים גם ב-**Explain Analyze:**

בשאילתא הראשונה:

EXPLAIN
-> Filter: (actors_no_index.id = 12345) (cost=82457 rows=81565) (actual time=6.27..327 rows=1 loops=1) -> Table scan ...

בשאילתא השנייה:

EXPLAIN
-> Filter: (md5(actors.id) = <cache>(md5(12345))) (cost=82192 rows=816387) (actual time=21.6..914 rows=1 loops=1) -> Table scar

ניתן לראות שהעלות ריצה היא כמעט זהה,מכיוון שאין אינדקסים ולכן למערכת אין דרך "לקצר את השאילתא" ניתן לראות שהערך אינו הפיך בקלות, ולכן הריצה השנייה הייתה איטית יותר.

מה לגבי שימוש ללא אינדקסים?

ניצור כעת טבלה ללא אינדקסים:

```
CREATE TABLE actors_no_index AS
SELECT *
FROM actors
```

אותה שליפה:

```
SELECT *
FROM actors_no_index
WHERE id = 12345
```

כעת לוקחת בממוצע 3.2 שניות ! (ממוצע של 5 שליפות - בדרך כלל כדי למדוד ביצועים נהוג למדוד יותר, אך מדובר בדוגמה). ניתן לראות שאותה השליפה לא משתמשת באינדקסים באמצעות Explain Analyze:

EXPLAIN
-> Filter: (actors_no_index.id = 12345) (cost=82457 rows=81565) (actual time=11.5..342 rows=1 loops=1) -> Table scan ...

נבדוק כעת גם את השליפה השנייה על אותה הטבלה ללא האינדקסים:

```
SELECT *
FROM roles_no_index
WHERE 7 * actor_id + 4 = 86419
```

השליפה הזו לוקחת בממוצע 3.5 שניות!, כלומר פי 6 מהשליפה הקודמת.

ניתן לראות באופן ברור שהאינדקסים משפרים את ביצועי השאילתא, זאת מכיוון שלמערכת מאוחסן מראש מיקומי ה-Actor_Id, המערכת קופצת ישירות לאזור שבו מאוחסנות ההופעות ומחזירה את התוצאה פי עשרות/מאות יותר מהר. (בשליפה הראשונה, האינדקסים הורידו את זמן השאילתא לכמעט 0.)

למה נרמול יכול לפגוע בביצועים?

נרמול משפר את ארגון הנתונים, מפחית כפילויות ומחלק מידע בין טבלאות כדי לשמור על עקביות.

אבל ככל שרמת הנרמול עולה, כך נדרשים יותר JOIN-ים כדי לאסוף את הנתונים המלאים.

פעולות JOIN רבות מאטות שאילתות, במיוחד על טבלאות גדולות, משום שהמערכת צריכה לשלב מידע ממספר מקורות.

בקריאות מרובות, לפעמים נרמול (שמירה על נתונים כפולים או מחושבים מראש) יכול לזרז שאילתות, על חשבון יותר שטח אחסון ותחזוקה מסובכת יותר.

איך אפשר לשפר את ביצועי השאילתות?

לא לשלוף מידע שאנחנו לא צריכים. זה חשוב מאוד עם שרת מרוחק כשהמידע עובר ברשת.

הימנעות מ--`SELECT *` עדיף לשלוף רק את העמודות הנחוצות.

הימנעות מ-`SELECT DISTINCT` -עדיף לסנן נכון מראש במקום להסיר כפילויות בדיעבד.

בשימוש עם `JOIN` עדיף להשתמש ב-(INNER/LEFT) - קריא וברוב המקרים גם יעיל יותר.

סינון מוקדם עם `WHERE` - עדיף מאשר לחכות ל-`HAVING` אחרי אגרגציה. (Group by) אם אפשר לסנן בתוך Subquery או בתנאי Join גם עדיף.

שימוש ב-`LIMIT` כשבודקים/מדגימים -כדי להקטין עומס.

בסביבת עבודה משותפת, כדאי להריץ שאילתות כבדות בשעות ללא הרבה עבודה, כדי לא להפריע למשתמשים אחרים.

שימוש ב-Join מועדף יותר משימוש ב-In.

דוגמאות להשפעות אלו נמצאות בעמוד הבא:

דוגמאות:

1. דוגמה להשפעות של הגבלת תוצאות + בחירת עמודות על הביצועים:

או אם נניח אנחנו רוצים את ה-1000 סרטים העדכניים ביותר:

אפשרות ראשונה היא לשלוף את כל הטבלה, ולסדר אותה בסדר יורד כך שה-1000 הראשונים יהיו החדשים ביותר.

```
select * from movies
order by year desc
```

שליפה כזו, לוקחת בממוצע 0.5 שניות, לעומת אם מגבילים את התוצאות ל-1000.

```
select * from movies
order by year desc
limit 1000
```

שליפה כזו לוקחת בממוצע 0.25 שניות.

מה לגבי שליפה של כל הרשומות בטבלה לעומת עמודה אחת? נניח ו אנחנו רוצים רק את ה-IDs, ולא רלוונטי לנו השנים או שם הסרט, כי אנחנו רק רוצים רשימה של ה-1000 העדכניים ביותר:

```
select id from movies
order by year desc
limit 1000
```

השליפה מתקצרת ל-0.06 שניות! לעומת שליפה של 0.5 שניות (קצרה פי 10) !

אם לא נמיין, אלא רק נשלוף את ה-1000 הראשונים

```
select id from movies
1000 limit
```

השליפה מתקצרת ל-0.02 שניות!

2. דוגמה שממחישה איך הביצועים משתנים בין **Distinct** לבין **GroupBy** למקרה שצריך לסנן?

נניח שאנחנו רוצים לראות את כל הזוגות של סרטים ושחקנים יחודיים::

נשלוף באמצעות Distinct:

```
SELECT DISTINCT movie_id, actor_id
FROM roles;
```

השליפה לוקחת בממוצע 0.6 שניות,

שימוש ב-Group By:

```
SELECT DISTINCT movie_id, actor_id
FROM roles;
```

השליפה מהירה במעט, לוקחת בממוצע 0.4 שניות,

אם נשתמש ב-Explain כדי לבדוק מה השוני בין השאילתות, שני השאילתות יחזירו תוצאה זהה! כלומר בשימוש ב-Group By ו-Distinct המערכת פועלת במנגנון זהה:

```
EXPLAIN SELECT DISTINCT actor_id FROM roles;
EXPLAIN SELECT actor_id FROM roles GROUP BY actor_id;
```

שתי השאילתות יחזירו את אותה התוצאה:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	roles	NULL	index	PRIMARY,actor_id,movie_id	actor_id	4	NULL	3200348	100.00	Using index

כלומר במקרה של שאילתא כזו, אין הבדל במנגנון שליפה של המערכת.

לעיתים בטבלאות גדולות יותר, מכיוון שהמנוע צריך להשוות יותר עמודות ייתכן ש-GroupBy יירוץ במעט מהיר יותר.

3. דוגמה שממחישה למה עדיף להשתמש ב-JOIN ולא ב-IN. נניח ואנחנו רוצים את כל הסרטים שהבמאים שלהם נכללים בטבלת הבמאים, שאילתא ראשונה בשימוש עם JOIN:

```
SELECT m.name
FROM movies m
JOIN movies_directors md ON m.id = md.movie_id
;JOIN directors d        ON d.id = md.director_id
```

השאילתא לוקחת בממוצע 0.01 שניות, לעומת שימוש ב-IN:

```
SELECT m.name
FROM movies m
WHERE m.id IN (
   SELECT md.movie_id
   FROM movies_directors md
);
```

שלוקח בממוצע 0.4 שניות, פי 40 יותר זמן!

השאילתה עם **JOIN** רצה מהר בהרבה כי מנוע ה-SQL יודע לבצע חיבור ישיר בין הטבלאות תוך שימוש באינדקסים, ומחזיר את הנתונים בצורה יעילה. לעומת זאת, בשימוש ב-**IN** המערכת נאלצת לבדוק לכל סרט האם הוא נמצא ברשימת מזהי הסרטים מתת־שאילתה, מה שגורם לסריקה חוזרת ופחות אופטימלית ולכן לוקח בממוצע פי 40 יותר זמן.