

Informations- arkitektur och databasutveckling

Mikael Olsson
v14 - Torsdag





Socrative

<https://www.socrative.com/>

- Frågehanterare
 - Logga in som student
 - Ange rum “Emmio”
 - Få upp en vänta-skärm



Waiting for the next activity to begin...



Transaktioner

Account		
AccountID	Name	Amount
10	Nils	3100
20	Stina	2500



Transaktioner

- Stina vill överföra 200:- till Nils.
 - `UPDATE Account SET Amount = Amount + 200 WHERE AccountID = 10;`
 - `UPDATE Account SET Amount = Amount - 200 WHERE AccountID = 20;`
- Vad händer om systemet kraschar mellan dessa uppdateringar?
- Hur kan vi bibehålla dataintegriteten?



Transaktioner

```
START TRANSACTION;
```

```
UPDATE Account SET Amount = Amount + 200 WHERE  
AccountID = 10;
```

```
UPDATE Account SET Amount = Amount - 200 WHERE  
AccountID = 20;
```

```
COMMIT;
```



Transaktionen

- AUTOCOMMIT



Transaktioner

- Ex i PHP, C# är troligen liknande

```
mysql_query("SET AUTOCOMMIT=0");  
mysql_query("START TRANSACTION");  
$result = mysql_query("UPDATE Account SET [...]");  
$result = $result && mysql_query("UPDATE Account [...]");  
if($result) {  
    mysql_query("COMMIT");  
} else {  
    mysql_query("ROLLBACK");  
}
```



Prestanda

```
CREATE TABLE employee (  
    employee_number char(10) NOT NULL,  
    firstname varchar(40),  
    surname varchar(40),  
    address text,  
    tel_no varchar(25),  
    salary int(11),  
    overtime_rate int(10) NOT NULL  
);
```




Prestanda

För att hitta Mary Pattersons lön (anställningsnr 1056) skulle vi köra något liknande följande:

```
SELECT salary  
FROM employee  
WHERE employee_number = '1056';
```



Prestanda

- Databasmotorn har ingen aning om var den ska hitta denna post.
- Den vet inte ens att det bara finns en matchning (employee_number är ju tabellens primärnyckel), så även om den hittar en matchande post så fortsätter den att leta igenom hela tabellen.



Prestanda - index

- Ett index är ett separat arkiv som enbart innehåller det/de fält du är intresserad av att sortera på.
- Om du skapar ett index på `employee_number` kan MySQL hitta motsvarande post väldigt snabbt.
- Index fungerar väldigt likt ett register i en bok.



Prestanda - EXPLAIN

- EXPLAIN visar (förklarar) hur dina frågor används.
- Genom att sätta EXPLAIN före en SELECT kan du se huruvida index används på rätt sätt och vilken typ av JOIN som utförs.

```
1 • EXPLAIN
2 SELECT employee_number, firstname, surname
3 FROM employee
4 WHERE employee_number= '1056';
```

Overview Output Snippets Result (1) ×

Fetches 1 records. Duration: 1.404 sec, fetched in: 0.031 sec

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	employee	ALL	NULL	NULL	NULL	NULL	23	Using where



Prestanda - EXPLAIN

- *table* visar vilken tabell det handlar om (om man har flera)

```
1 • EXPLAIN
2 SELECT employee_number, firstname, surname
3 FROM employee
4 WHERE employee_number= '1056';
```

Overview Output Snippets Result (1) ×

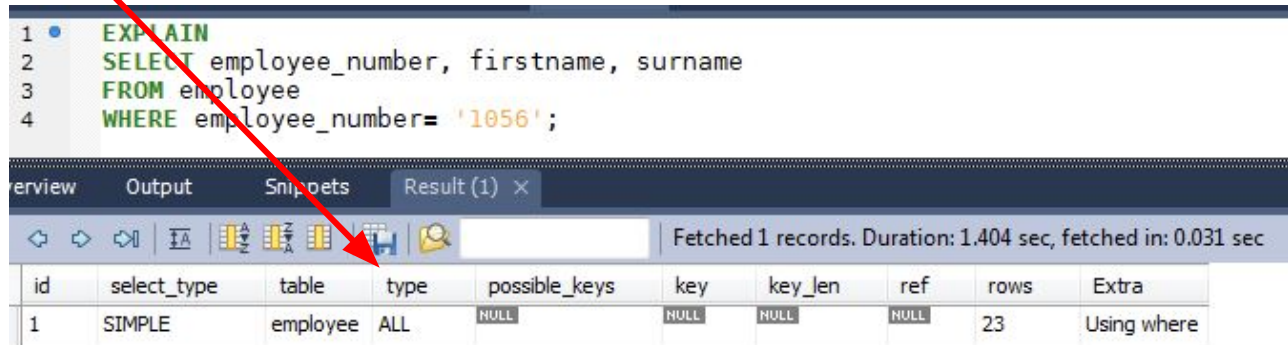
Fetches 1 records. Duration: 1.404 sec, fetched in: 0.031 sec

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	employee	ALL	NULL	NULL	NULL	NULL	23	Using where

Prestanda - EXPLAIN

- *type* är viktig – den visar vilken typ av JOIN som används. Från bäst till sämst är de möjliga värdena följande:

- system
- const
- eq_ref
- ref
- range
- index
- all



```
1 • EXPLAIN
2 SELECT employee_number, firstname, surname
3 FROM employee
4 WHERE employee_number= '1056';
```

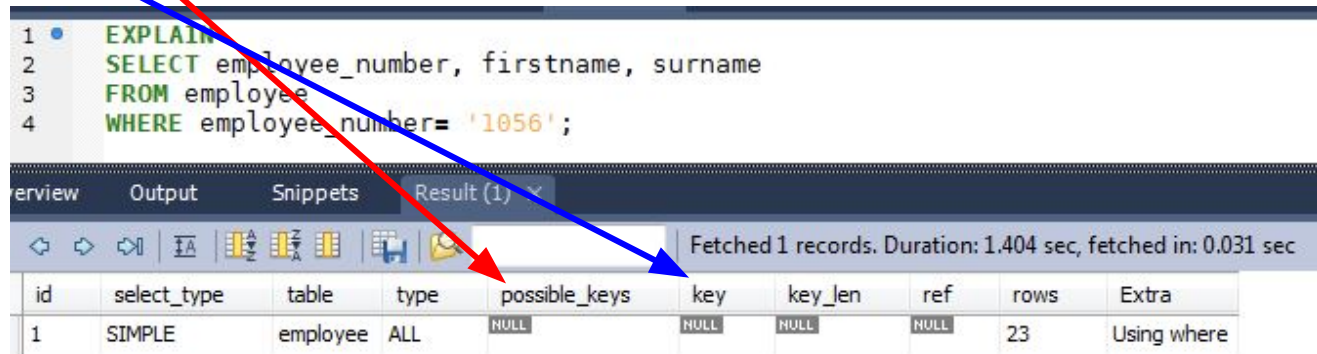
Overview Output Snippets Result (1) ×

Fetches 1 records. Duration: 1.404 sec, fetched in: 0.031 sec

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	employee	ALL	NULL	NULL	NULL	NULL	23	Using where

Prestanda - EXPLAIN

- *possible_keys* visar vilka möjliga index som gäller för denna tabell
- *key* ... och vilken som faktiskt används.



```
1 EXPLAIN
2 SELECT employee_number, firstname, surname
3 FROM employee
4 WHERE employee_number = '1056';
```

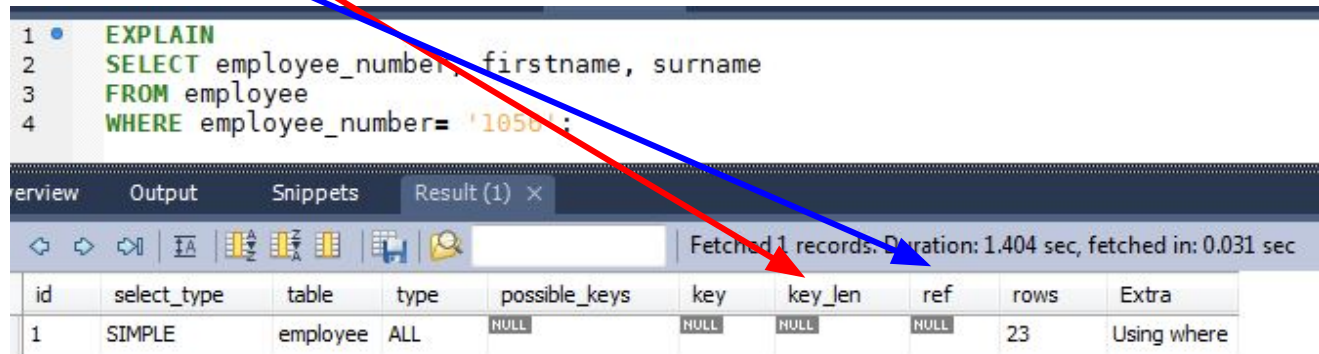
Overview Output Snippets Result (1) x

Fetches 1 records. Duration: 1.404 sec, fetched in: 0.031 sec

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	employee	ALL	NULL	NULL	NULL	NULL	23	Using where

Prestanda - EXPLAIN

- *key_Len* ger oss längden på nyckeln. Ju kortare desto bättre.
- *ref* säger vilken kolumn, eller konstant, som används.



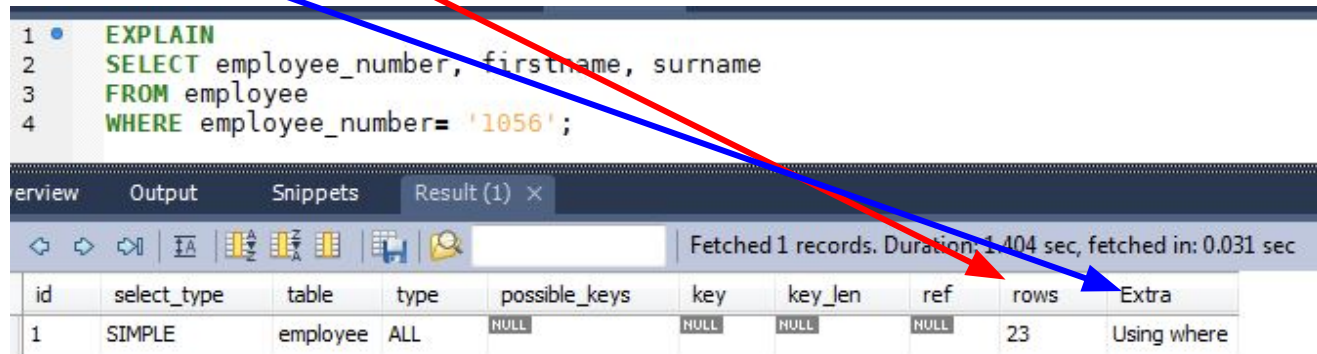
```
1 EXPLAIN
2 SELECT employee_number, firstname, surname
3 FROM employee
4 WHERE employee_number = '1050';
```

Fetches 1 records. Duration: 1.404 sec, fetched in: 0.031 sec

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	employee	ALL	NULL	NULL	NULL	NULL	23	Using where

Prestanda - EXPLAIN

- *rows* – antal rader som MySQL tror sig behöva undersöka för att få tag i rätt data.
- *extra* - extra info – Här vill man helst inte se "using temporary" eller "using filesort".



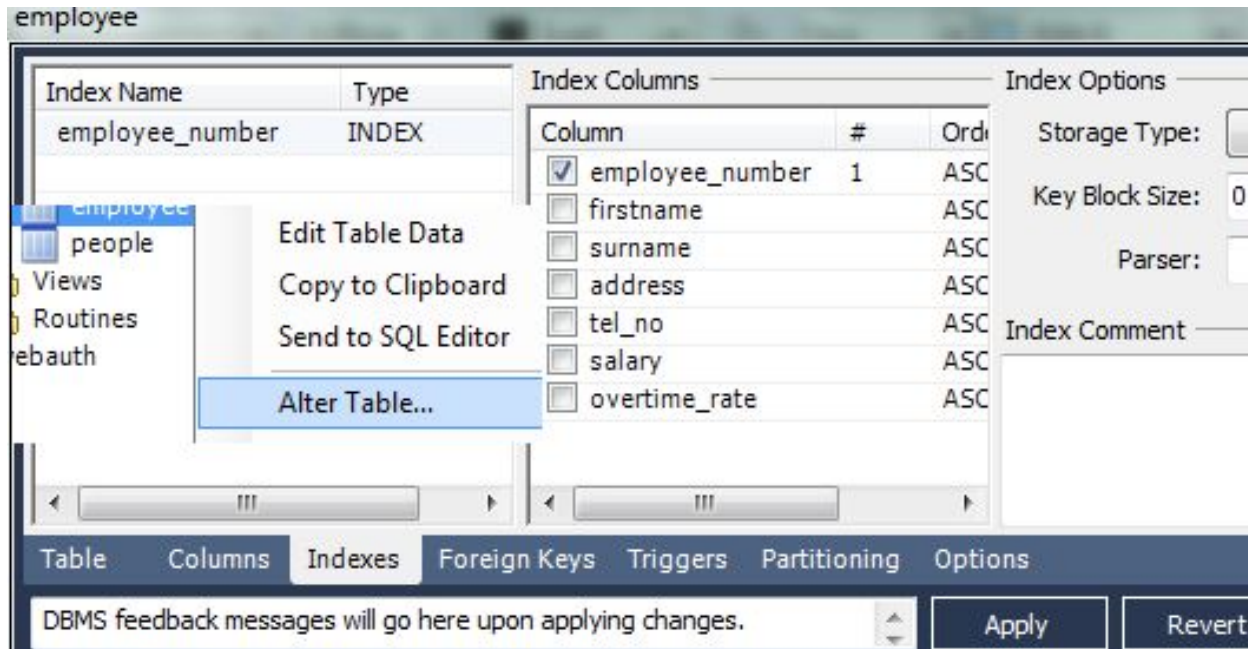
```
1 EXPLAIN
2 SELECT employee_number, firstname, surname
3 FROM employee
4 WHERE employee_number= '1056';
```

Fetches 1 records. Duration: 1.404 sec, fetched in: 0.031 sec

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	employee	ALL	NULL	NULL	NULL	NULL	23	Using where

Prestanda - index

- Vi lägger till indexet vi pratade om tidigare.



Prestanda - index

- Mycket bättre! "JOIN"-typen är nu const, vilket betyder att tabellen bara har en matchande rad.
- Primärnyckeln används för att hitta post.
- Antalet rader som MySQL tror sig behöva leta igenom är 1.

```
1  EXPLAIN
2  SELECT employee_number, firstname, surname
3  FROM employee
4  WHERE employee_number= '1056';
```

Overview Output Snippets Result (1) x

Fetches 1 records. Duration: 0.093 sec, fetched in: 0.000 sec

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	employee	ref	employee_number	employee_number	30	const	1	Using where



Prestanda - index

- Nackdelar med index
 - När du uppdaterar en tabell måste databasen uppdatera indexet också, så det finns ett prestandapris att betala för index.
 - Om ditt system inte kör många fler INSERT än SELECT och INSERTerna måste vara snabba och inte SELECTerna är detta ett pris värt att betala.



Prestanda - index

- Okej, men om du vill köra en SELECT på mer än ett kriterium?
- Det är bara vettigt att indexera de fält som du använder i WHERE-satsen.
- `SELECT firstname FROM employee;`
använder inget index. Ett index på *firstname* är värdelöst.
- Men. `SELECT firstname FROM employee WHERE surname="Madida";`
skulle ha nytta av ett index på *surname*.



Prestanda - index

- Ett lite knepigare exempel.
- Vi vill hitta alla anställda vars halva övertidsersättning är mindre än \$20.
- Vilken kolumn ska vi lägga ett index på?
 - Rimligtvis *overtime_rate*
- Varför?
 - Det är den kolumnen vi ska använda i *WHERE*-satsen.



Prestanda - index

- `ALTER TABLE employee ADD INDEX(overtime_rate);`

```
1 • EXPLAIN
2   SELECT firstname
3   FROM employee
4   WHERE overtime_rate/2<20;
```

Result (1) x									
Overview Output Snippets									
Fetches 1 records. Duration: 0.094 sec, fetched in: 0.000									
id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	employee	ALL	NULL	NULL	NULL	NULL	23	Using where

Ser det bra ut?



Prestanda - index

- Vi kan skriva om villkoret:

Istället för

```
WHERE overtime_rate / 2 < 20
```

kan vi skriva

```
WHERE overtime_rate < 20 * 2
```




Prestanda - index

- MySQL kan göra beräkningen $2*20$ en gång och sedan använda den som en konstant att leta efter i indexet.

```
1 • EXPLAIN
2 SELECT firstname
3 FROM employee
4 WHERE overtime_rate < 20*2;
```

Overview

Output

Snippets

Result (1) x



Fetches 1 records. Duration: 0.031 sec, fetched in: 0.000 sec

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	employee	range	overtime_rate	overtime_rate	4	NULL	1	Using where



Prestanda - index

- Att sortera på efternamn är ett vanligt krav, så det verkar rimligt att skapa ett index på *surname*.
- Vi kan dock tänka oss att vi har tusentals anställda med efternamnet "Smith".
- Då måste vi indexera *firstname* också.
- MySQL använder *leftmost prefixing* (prefixet längst till vänster), vilket betyder att ett index på flera kolumner A, B, C inte enbart används för att söka på A, B, C-kombinationer utan även A, B eller enbart A.



Prestanda - index

- I vårt exempel betyder det att
- `ALTER TABLE employee ADD INDEX(surname,firstname);`
- används för frågor som t ex
- `EXPLAIN SELECT overtime_rate FROM employee WHERE surname='Madida';`
- men även
- `EXPLAIN SELECT overtime_rate FROM employee WHERE surname='Madida' and firstname="Mpho";`



Prestanda - index

Båda resulterar i detta.

```
1 • EXPLAIN
2 SELECT overtime_rate
3 FROM employee
4 WHERE surname='Madida';
```

Result (1) X									
Overview Output Snippets									
Fetches 1 records. Duration: 0.000 sec, fetched in: 0.000									
id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	employee	ref	surname	surname	123	const	1	Using where



Prestanda - index

- Emellertid använder inte följande fråga ett index.
 - `EXPLAIN SELECT overtime_rate FROM employee WHERE firstname='Mpho';`
- Varför inte?
 - *firstname* är inte tillgängligt från vänster i indexet.



Prestanda - index

- Om man behöver en sån fråga får man lägga ett separat index för *firstname*.

```
1 • EXPLAIN
2 SELECT overtime_rate
3 FROM employee
4 WHERE firstname='Mpho';
```

Overview

Output

Snippets

Result (1) ×

Navigation icons: back, forward, search, etc. | Fetched 1 records. Duration: 0.000 sec, fetched in: 0.000

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	employee	ALL	NULL	NULL	NULL	NULL	23	Using where



Prestanda - query optimizer

- Hur går det till när databasen väljer vilka nycklar som ska användas i en fråga?
- Processen kallas *query optimizer*, frågeoptimeraren.
- Den tar först en snabbkoll på indexet för att se vilka index som är lämpligast att använda.



Prestanda - query optimizer

- Jämför med att leta efter en cd, "Brutal Planet" av "Alice Cooper".
 - Vi har två index.
 - Alfabetisk över artistnamn.
 - Alfabetisk över skivnamn.
 - En snabbkoll säger att vi har 20.000 unika artister och 400.000 unika album, så vi bestämmer oss för att leta efter artist.



Prestanda - query optimizer

- Om du visste att det finns 20 Alice Cooper-skivor och att "Brutal Planet" är det enda albumet som startar med "B", så skulle sökkriterierna ändras.
- Du kan förse optimeraren med liknande information genom att köra
`ANALYZE TABLE tabellnamn;`
- Detta sparar nyckeldistributionen för en tabell.



Prestanda - optimize table

- Många DELETE och UPDATE lämnar “luckor” i tabellen/filsystemet, speciellt när du använder varchar, eller text/blob-fält.
- Detta betyder fler onödiga läs/skriv-accesser till hårddisken.
- `OPTIMIZE TABLE tabellnamn;`
löser detta problem.



Prestanda - index

- Man måste inte indexera hela fält.
 - Våra fält *surname* och *firstname* är 40 tecken var.
 - Det betyder att indexet vi skapade för dessa fält är 80 tecken.
 - INSERT till denna tabell måste då också skriva 80 extra tecken och SELECT har 80 extra tecken att manövrera runt.



Prestanda - index

- Det är onödigt att ha CHAR (255) för surname och firstname om namn aldrig är längre än 20 tecken.
- Man vill inte "skära av" namn, men man kan alltid ändra storlek om förutsättningarna ändras.
- Använd VARCHAR istället för CHAR.
 - En del rekommenderar inte detta eftersom det kan leda till mer fragmentation, men det går att komma runt genom att använda OPTIMIZE ofta.



MySQL vs MS SQL Server

- Olika databaser har olika funktioner.
 - MySQL fick t ex inte stöd för Stored Procedures eller Triggers förrän i versioner runt 5.0.
- Det som skiljer olika SQL-dialekter åt är oftast olika funktioner eller olika namn för samma funktioner.
 - MS SQL = T-SQL
 - MySQL = SQL/PSM (inte lika vanlig term)



MySQL vs MS SQL Server

- Följande MySQL-funktioner är exempel som inte har stöd eller är svåra att replikera i T-SQL.
 - DATE_ADD med INTERVAL, DATE_SUB med INTERVAL, GET_FORMAT, PERIOD_ADD, PERIOD_DIFF, SUBTIME, TIMESTAMP, TIMESTAMPADD, TIMESTAMPDIF, MATCH



MySQL vs MS SQL Server

- MySQL: `LIMIT x, x`
- T_SQL: `SELECT TOP 10 firstName FROM Employees ORDER BY firstName OFFSET 10 ROWS;`



MySQL vs MS SQL Server

- MySQL har stöd för IF, T-SQL har det inte.
- MySQL: `IF(@a > @b , @a, @b - @a)`
- Istället kan man använda CASE i T-SQL:
`CASE WHEN @a > @b THEN @a ELSE @b - @a END`



MySQL vs MS SQL Server

- Slå ihop strängar:
 - MySQL:
 - `CONCAT('a', 'b', 'c')`
 - T-SQL:
 - `'a' + 'b' + 'c'`



MySQL vs MS SQL Server

- Datum-hantering, exempel dagens datum:
 - T-SQL: `GetDate()`
 - MySQL: `CURRDATE()` eller `NOW()`



MySQL vs MS SQL Server

- MySQL har en “pluggable” arkitektur för att kunna använda olika Storage Engines.
- MS SQL har enbart den inbyggda.

Lunch!





Repetition

- Frågor kring något?



Tack!

