Department of Computer Engineering

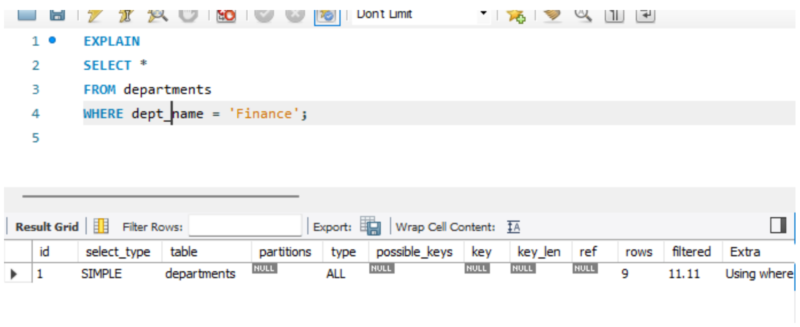
University of Peradeniya

CO527 Advanced Database Systems

Edirimanna Y.H.

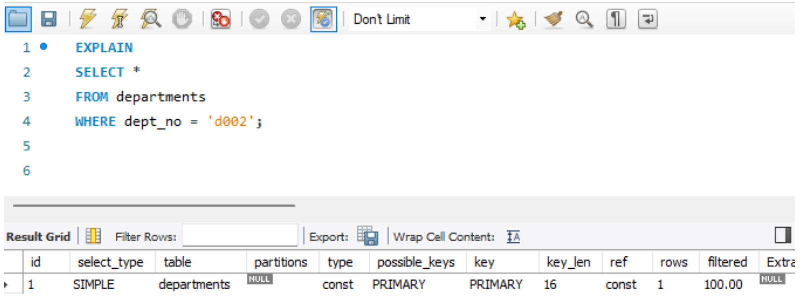
E/20/089

1) Troubleshooting Simple Queries Using EXPLAIN



```sql
EXPLAIN
SELECT *
FROM departments
WHERE dept_name = 'Finance';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	departments	NULL	ALL	NULL	NULL	NULL	NULL	9	11.11	Using where



```sql
EXPLAIN
SELECT *
FROM departments
WHERE dept_no = 'd002';
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	departments	NULL	const	PRIMARY	PRIMARY	16	const	1	100.00	NULL

Query 1: dept_name = 'Finance'

- key = NULL → No index is used
- type = ALL → Full table scan
- rows → MySQL checks all rows in departments
- Reason: dept_name is not indexed

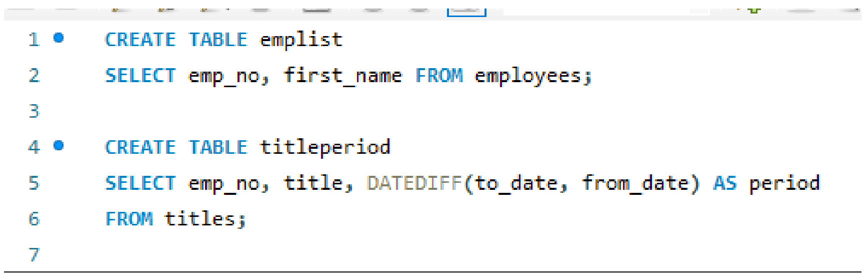
Interpretation: MySQL scans the entire department's table to find matching records, which is inefficient and does not scale well for large datasets.

Query 2: dept_no = 'd002'

- key = PRIMARY → Primary key index is used
- type = const → Fastest possible access type
- rows = 1 → Only one row examined
- Reason: dept_no is the primary key

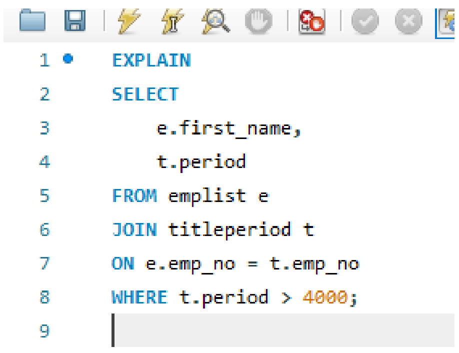
Interpretation: MySQL directly locates the required row using the primary key index, resulting in minimal row access and optimal performance.

2) Join Query Analysis Using EXPLAIN



```sql
CREATE TABLE emplist
SELECT emp_no, first_name FROM employees;

CREATE TABLE titleperiod
SELECT emp_no, title, DATEDIFF(to_date, from_date) AS period
FROM titles;

```

Initial Query



```sql
EXPLAIN
SELECT
    e.first_name,
    t.period
FROM emplist e
JOIN titleperiod t
ON e.emp_no = t.emp_no
WHERE t.period > 4000;

```

EXPLAIN Output Analysis

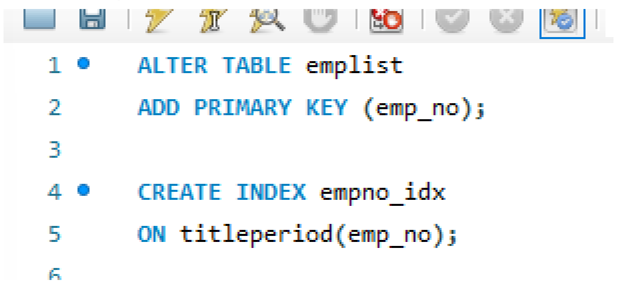
- Join type: ALL (full table scan on both tables)
- Rows examined: emplist_rows × titleperiod_rows
- MySQL uses a nested loop join
- Large number of row combinations must be checked
- No indexes available to speed up matching

Estimated Row Combinations

- emplist has 300,024 rows
- titleperiod has 443308 rows

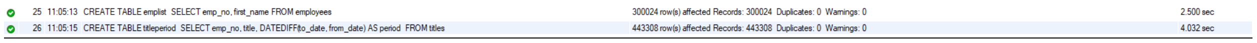
Then MySQL may evaluate hundreds of millions of row combinations, leading to poor performance. his is why the query is extremely slow.

3) <u>Indexing for Optimization</u>



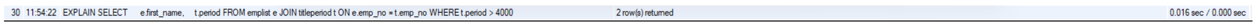
```sql
ALTER TABLE emplist
ADD PRIMARY KEY (emp_no);

CREATE INDEX empno_idx
ON titleperiod(emp_no);
```

Before Indexing



```
25  11:05:13  CREATE TABLE emplist  SELECT emp_no, first_name FROM employees    300024 row(s) affected Records: 300024  Duplicates: 0  Warnings: 0    2.500 sec
26  11:05:15  CREATE TABLE titleperiod  SELECT emp_no, title, DATEDIFF(to_date, from_date) AS period  FROM titles    443308 row(s) affected Records: 443308  Duplicates: 0  Warnings: 0    4.032 sec
```

After Indexing

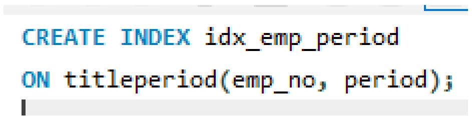


```
30  11:54:22  EXPLAIN SELECT    e.first_name,    t.period FROM emplist e JOIN titleperiod t ON e.emp_no = t.emp_no WHERE t.period > 4000    2 row(s) returned    0.016 sec / 0.000 sec
```

<u>The Improvement</u>

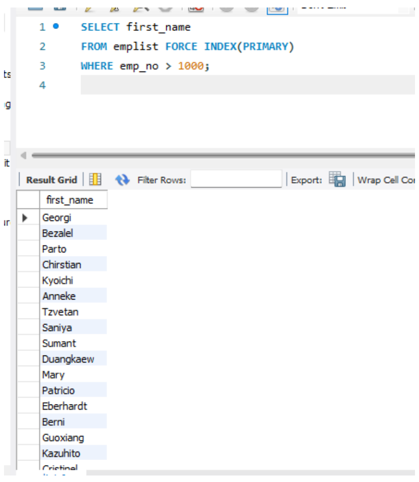
<u>Before Indexing</u>	**<u>After Indexing</u>**
Full table scans	Index lookups
Huge row combinations	Drastically reduced
key = NULL	key = PRIMARY / idx_emp_no
Slow	Fast

Is it possible to optimize the query execution further?
Yes it is possible to optimize. We can use Composite Index for this. This allows efficient join and efficient filtering.



```sql
CREATE INDEX idx_emp_period
ON titleperiod(emp_no, period);
```

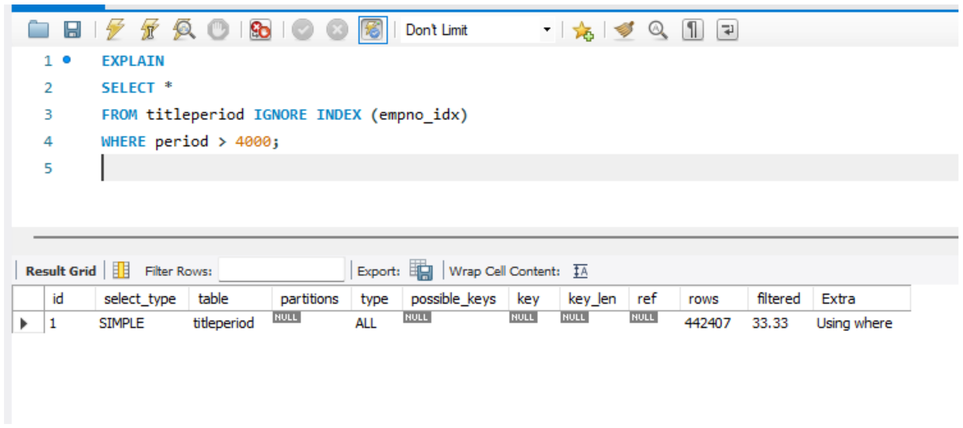
4) Query Rewriting Techniques
FORCE INDEX



```sql
SELECT first_name
FROM emplist FORCE INDEX(PRIMARY)
WHERE emp_no > 1000;
```

first_name
Georgi
Bezalel
Parto
Chirstian
Kyoichi
Anneke
Tzvetan
Saniya
Sumant
Duangkaew
Mary
Patricio
Eberhardt
Berni
Guoxiang
Kazuhito

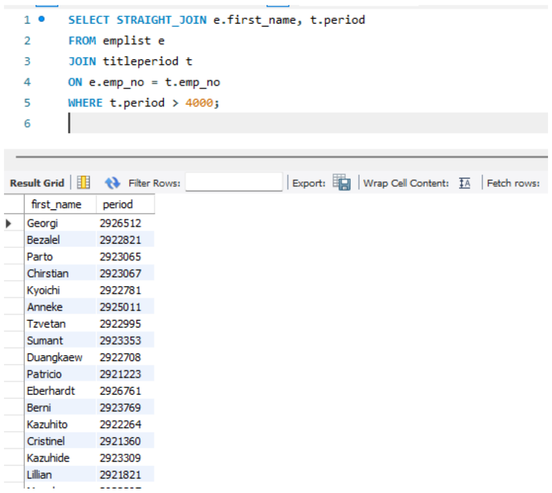
IGNORE INDEX



```sql
EXPLAIN
SELECT *
FROM titleperiod IGNORE INDEX (empno_idx)
WHERE period > 4000;
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	titleperiod	NULL	ALL	NULL	NULL	NULL	NULL	442407	33.33	Using where

STRAIGHT_JOIN



```sql
SELECT STRAIGHT_JOIN e.first_name, t.period
FROM emplist e
JOIN titleperiod t
ON e.emp_no = t.emp_no
WHERE t.period > 4000;
```

first_name	period
Georgi	2926512
Bezalel	2922821
Parto	2923065
Chirstian	2923067
Kyoichi	2922781
Anneke	2925011
Tzvetan	2922995
Sumant	2923353
Duangkaew	2922708
Patricio	2921223
Eberhardt	2926761
Berni	2923769
Kazuhito	2922264
Cristinel	2921360
Kazuhide	2923309
Lillian	2921821

These techniques provide **manual control** when the optimizer’s choice is suboptimal.