MySQL Gap Lock

Gap Lock이란?

```sql
use gap_lock;

CREATE TABLE tb_gaplock(
    id INT NOT NULL,
    name VARCHAR(50) DEFAULT NULL,
    PRIMARY KEY(ID)
);


INSERT INTO tb_gaplock
VALUES (1,'MAtt'),(3,'Esther'),(6,'Peter');
```

	id	name
▶	1	MAtt
	3	Esther
	6	Peter
*	NULL	NULL

레코드 없는 빈 공간(ID=2)

레코드 없는 빈 공간(ID=4,5)

레코드 사이의 간격 Lock

→ Gap Lock

Gap Lock이 왜 필요할까?

Session - 1

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.01 sec)

mysql> UPDATE tb_gaplock SET name='Updated' WHERE id=2;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 0  Changed: 0  Warnings: 0
```

Session -2

```
mysql> use gap_lock;
Database changed
mysql> SELECT * FROM performance_schema.data_locks;
```

트랜잭션을 실행 후 COMMIT하지 않은 상태에서 실행해보기

Gap Lock이 왜 필요할까?

ENGINE	ENGINE_LOCK_ID	ENGINE_TRANSACTION_ID	THREAD_ID	EVENT_ID	OBJECT_SCHEMA	OBJECT_NAME
INNODB	2821379399056:156:1091:2821369303576	8791	68	8	gap_lock	tb_gaplock
INNODB	2821379399056:156:29:4:3:2821367994392	8791	68	8	gap_lock	tb_gaplock

PARTITION_NAME	SUBPARTITION_NAME	INDEX_NAME	OBJECT_INSTANCE_BEGIN	LOCK_TYPE	LOCK_MODE	LOCK_STATUS	LOCK_DATA
NULL	NULL	NULL	2821369303576	TABLE	IX	GRANTED	NULL
NULL	NULL	PRIMARY	2821367994392	RECORD	X,GAP	GRANTED	3

X, REC_NOT_GAP → Exclusive Record Lock(해당 레코드만)
X, GAP → Exclusive Gap Lock(레코드 사이 갭)
X → Exclusive Next_Key Lock(레코드+사이 갭)
S, REC_NOT_GAP → Shaerd Gap Lock(해당 레코드만(읽기))

LOCK_TYPE=RECORD → 레코드 수준의 잠금

만약 INT(UNIQUE)가 아닌
DOUBLE,DATETIME(Non-Unique)TYPE 이였다면??

Gap Lock의 필요성

MySQL 서버의 Gap Lock은 크게 3가지 목적을 위해서 사용됨

1. Repeatable Read 격리 수준 보장
2. Replication 일관성 보장 (Binary Log Format = Statement 또는 Mixed)
3. Foreign Key 일관성 보장

Repeatable Read에서 Gap Lock이 뭐가 연관되어 있는데?

Gap Lock의 필요성

	Session-1	Session-2
1	SET transaction_isolation='READ-COMMITTED';	SET transaction_isolation='READ-COMMITTED';
2	BEGIN;	BEGIN;
3	SELECT * FROM tb_gaplock WHERE id BETWEEN 1 AND 3 FOR UPDATE;	
4		INSERT INTO tb_gaplock VALUES (2, 'Matt2');
5		COMMIT;
6	SELECT * FROM tb_gaplock WHERE id BETWEEN 1 AND 3 FOR UPDATE;	

Read Commited 격리 수준에서 발생하는 문제를 찾아보자 !

Gap Lock의 필요성

Session-1

```
mysql> SET transaction_isolation='READ-COMMITTED';
Query OK, 0 rows affected (0.00 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM tb_gaplock WHERE id BETWEEN 1 AND 3 FOR UPDATE;
+----+------+
| id | name |
+----+------+
|  1 | MAtt |
+----+------+
1 row in set (0.00 sec)
```

Gap Lock의 필요성

Session-2

```
mysql> SET transaction_isolation='READ-COMMITTED';
Query OK, 0 rows affected (0.00 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO tb_gaplock VALUES (2, 'Matt2');
Query OK, 1 row affected (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)
```

Gap Lock의 필요성

Session-1

```
mysql> SELECT * FROM tb_gaplock WHERE id BETWEEN 1 AND 3 FOR UPDATE;
+----+------+
| id | name |
+----+------+
|  1 | MAtt |
+----+------+
1 row in set (0.00 sec)

mysql> SELECT * FROM tb_gaplock WHERE id BETWEEN 1 AND 3 FOR UPDATE;
+----+-------+
| id | name  |
+----+-------+
|  1 | MAtt  |
|  2 | Matt2 |
+----+-------+
2 rows in set (0.00 sec)
```

```
+-------------+----------------+-
| LOCK_TYPE   | LOCK_MODE      |
+-------------+----------------+-
| TABLE       | IX             |
| RECORD      | X,REC_NOT_GAP  |
| RECORD      | X,REC_NOT_GAP  |
```

Read Committed 수준에서는 Gap Lock이 걸리지 않음 →Phantom Read 발생

Gap Lock의 필요성

	Session-1	Session-2
1	SET binlog_format='STATEMENT';	SET binlog_format='STATEMENT';
2	BEGIN;	BEGIN;
3	UPDATE tb_gaplock SET name='Dummy' WHERE id BETWEEN 1 AND 3;	
4		INSERT INTO tb_gaplock VALUES (2, 'Matt2');

바이너리 로그 포맷에 의한 Gap Lock 사용 여부 예제

Gap Lock의 필요성

Session-1

```
mysql> SET binlog_format='STATEMENT';
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE tb_gaplock SET name='DUMMY' WHERE id BETWEEN 1 AND 3
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Gap Lock의 필요성

Session-2

```
mysql> SET binlog_format='STATEMENT';
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO tb_gaplock VALUES (2,'Matt2');
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

Sessionn-1 을 COMMIT하고 다시 시도해보자

Gap Lock의 필요성

Session-1

```
mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)
```

Session-2

```
mysql> INSERT INTO tb_gaplock VALUES (2,'Matt2');
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
mysql> INSERT INTO tb_gaplock VALUES (2,'Matt2');
Query OK, 1 row affected (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)
```

Gap Lock 으로 Replication 일관성 보장 → 바이너리 로그를 통해 좀 더 확인해보자!

Gap Lock의 필요성

시나리오1: Gap Lock 있음

```
시간  Session-1 (STATEMENT)          Session-2 (STATEMENT)
──────────────────────────────────────────────────────────
T1    BEGIN
T2    UPDATE ... WHERE id BETWEEN 1 AND 3
      → id=1 레코드 락
      → (1, 4) Gap Lock 획득 🔒
T3                                   BEGIN
T4                                   INSERT (2, 'Matt2')
                                     → 블로킹! ⏳ (Gap Lock 때문)
T5    COMMIT ✅
      → Gap Lock 해제
T6                                   INSERT 완료!
T7                                   COMMIT ✅
```

Replica와 matser의
최종 상태는 일치!!

	id	name
▶	1	DUMMY
	2	Matt2
	4	SomeName
	6	Peter
*	NULL	NULL

바이너리 로그

```
1. UPDATE tb_gaplock SET name='Dummy' WHERE id BETWEEN 1 AND 3;
2. INSERT INTO tb_gaplock VALUES (2, 'Matt2');
```

Gap Lock의 필요성

시나리오2: Gap Lock 없음

```
시간  Session-1                          Session-2
──────────────────────────────────────────────────

T1    BEGIN
T2    UPDATE ... WHERE id BETWEEN 1 AND 3
      → id=1 레코드 락만
      → Gap Lock 없음 ❌
T3                                       BEGIN
T4                                       INSERT (2, 'Matt2')
                                         → 즉시 성공! ✅ (Gap Lock 없어서)
T5                                       COMMIT ✅ (먼저!)
T6    COMMIT ✅
```

Replica와 matser의
최종 상태는 불일치!!

```
id=1, name='Dummy'
id=2, name='Matt2'
```

<Master>

```
id=1, name='Dummy'
id=2, name='Dummy'
```

<Replica>

바이너리 로그-실행 순서 꼬임

```
1. INSERT INTO tb_gaplock VALUES (2, 'Matt2');  ← 먼저 COMMIT
2. UPDATE tb_gaplock SET name='Dummy' WHERE id BETWEEN 1 AND 3;  ← 나중 COMMIT
```

Gap Lock의 특징

1. Shared Gap Lock = Exclusive Gap Lock

2. Next Key Lock= Record Lock + Gap Lock

	id	name
	1	DUMMY
▶	4	SomeName
	6	Peter
*	NULL	NULL

Gap Lock의 특징 - Shared Gap Lock

Session-1 -- UPDATE

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE tb_gaplock SET name='A' WHERE id=2;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Session-2 -- DELETE

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE tb_gaplock SET name='B' WHERE id=3;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 0  Changed: 0  Warnings: 0
```

Session-3 SELECT .. FOR UPDATE

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> DELETE FROM tb_gaplock WHERE id=2;
Query OK, 0 rows affected (0.00 sec)
```

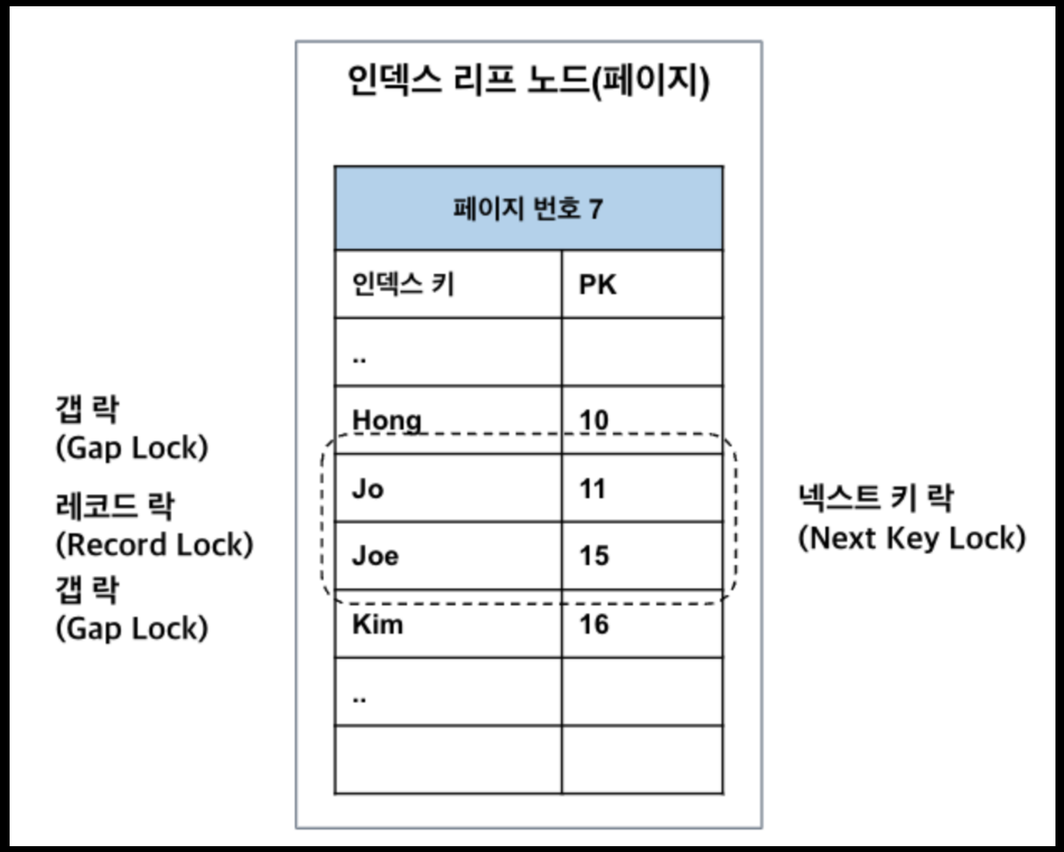
Session-4 -- UPDATE

```
mysql> INSERT INTO tb_gaplock VALUES (2, 'Test');
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

Gap Lock은 내부적으로 모두 Shared Lock

BUT, Phantom Read 방지 위해서 INSERT만 방지

Gap Lock의 특징 - Next Key Lock



Record Rock+ Gap Lock

Gap Lock 주의사항

	Session-1	Session-2
1	BEGIN;	BEGIN;
2	UPDATE tb_gaplock SET name='Dummy' WHERE id=5;	
3		INSERT INTO tb_gaplock VALUES (1, 'Matt');

테이블의 데이터를 다 지우고 다음 시나리오를 진행해보자

Gap Lock 주의사항

Session-1

```
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE tb_gaplock SET name='Dummy' WHERE id=5;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 0  Changed: 0  Warnings: 0
```

데이터가 없기 때문에 pk 시작 ~ 마지막 지점까지 간격을 잠궈버림

Session-2

```
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO tb_gaplock VALUES (1,'Matt');
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

어떠한 값이라도 INSERT 불가능함 → 레코드가 적을수록 Gap Lock의 간격 넓어지는 현상 확인 가능

Gap Lock과 Dead Lock

	Session-1	Session-2
1	BEGIN;	BEGIN;
2	UPDATE tb_gaplock SET ... WHERE id=1;	UPDATE tb_gaplock SET ... WHERE id=3;
3	UPDATE tb_gaplock SET ... WHERE id=3;	
4		UPDATE tb_gaplock SET ... WHERE id=1;

Dead Lock의 가장 단순한 형태

Gap Lock과 Dead Lock

Dead Lock의 복잡한 형태를 생각해보자

먼저, MySQL 서버에서 동등 비교 조건이 인덱스 종류별로 어떤 잠금을 이용하는지 알아보자 !

Gap Lock과 Dead Lock

Primary Key , Unique Key

쿼리의 조건이 1건의 결과를 보장

Record Lock

쿼리의 조건이 1건의 결과를 보장X

Record Lock+Gap Lock

Gap Lock과 Dead Lock

Primary Key , Non-Unique Key

쿼리의 결과 대상 레코드 건수에 상관없이 항상

Record Lock+Gap Lock

Gap Lock과 Dead Lock

Gap Lock

INSERT Intention Gap Lock

	Session-1	Session-2
1	BEGIN;	BEGIN;
2	SELECT * FROM tb_gaplock WHERE id=2 /* Not-Existed id */ FOR UPDATE;	SELECT * FROM tb_gaplock WHERE id=2 /* Not-Existed id */ FOR UPDATE;
3	DELETE FROM tb_gaplock WHERE id=2;	DELETE FROM tb_gaplock WHERE id=2;
4	INSERT INTO tb_gaplock VALUES (2, 'Matt2');	
5		INSERT INTO tb_gaplock VALUES (2, 'Matt2');

Gap Lock과 INSERT Intention Gap Lock의 충돌

Gap Lock과 Dead Lock

그래서 Dead Lock이 MySQL 서버의 오류라고 생각해?

DeadLock 해결법은 각자 스터디에서 하시길 ^_^