InnoDB 잠금 전략

Overview locking strategies in InnoDB

2025.08.20.

System Software Lab.

목차

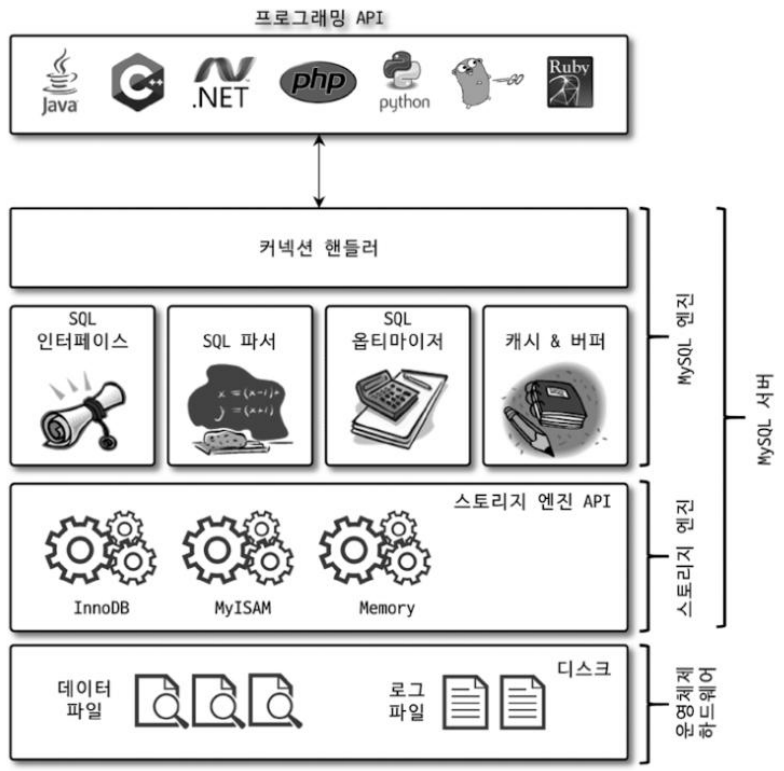
01

MySQL 엔진과 스토리지 엔진

MySQL 엔진과 스토리지 엔진

MySQL 엔진

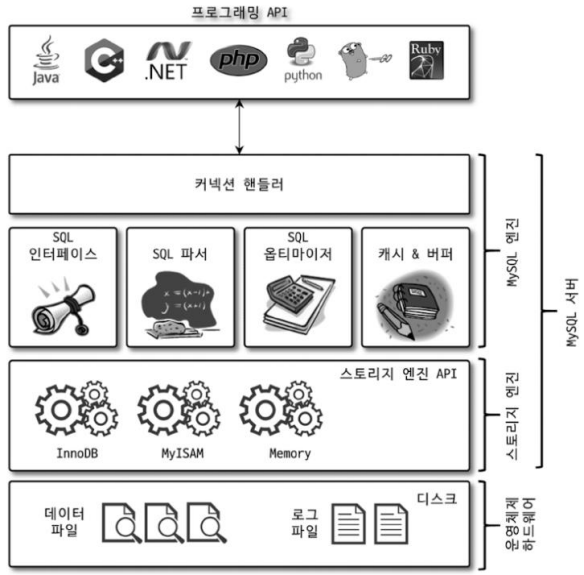
- 클라이언트의 접속 및 쿼리 요청을 처리, SQL문을 파싱하고 쿼리를 최적화



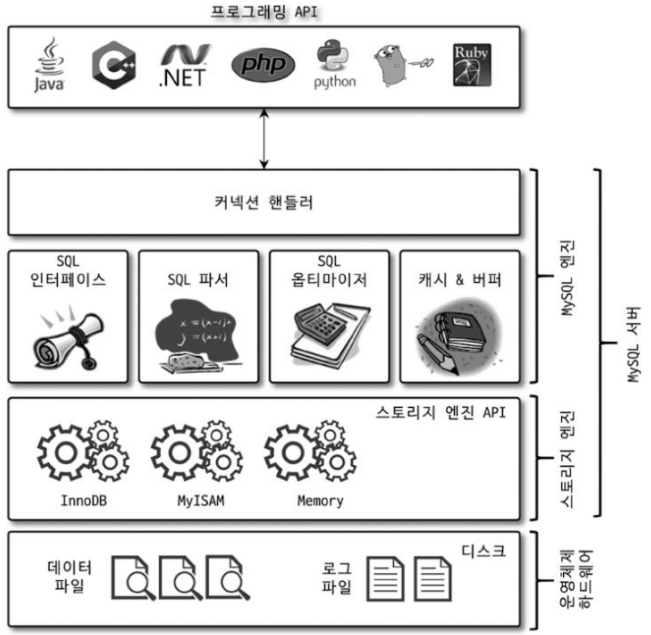
MySQL 엔진과 스토리지 엔진

스토리지 엔진

- 데이터를 디스크에 저장하거나 디스크에서 데이터를 읽음
- MySQL 엔진은 하나이지만 스토리지 엔진은 여러 개 존재하며 여러 개를 동시에 사용할 수 있음
- Blackhole, CSV, Archive, Memory, NDB Cluster, MyISAM, InnoDB 등 다양한 종류가 존재
- MyISAM을 기본 스토리지 엔진으로 하다가 MySQL 5.5 이상부터 InnoDB를 기본 스토리지 엔진으로 채택
- MyISAM은 트랜잭션을 지원하지 않지만 InnoDB는 트랜잭션을 지원한다는 점이 큰 차이점



MySQL 엔진과 스토리지 엔진



MySQL 엔진

- 클라이언트의 접속 및 쿼리 요청을 처리, SQL문을 파싱하고 쿼리를 최적화

스토리지 엔진

- 데이터를 디스크에 저장하거나 디스크에서 데이터를 읽음
- MySQL 엔진은 하나이지만 스토리지 엔진은 여러 개 존재하며 여러 개를 동시에 사용할 수 있음
- Blackhole, CSV, Archive, Memory, NDB Cluster, MyISAM, InnoDB 등 다양한 종류가 존재
- MyISAM을 기본 스토리지 엔진으로 하다가 MySQL 5.5 이상부터 InnoDB를 기본 스토리지 엔진으로 채택
- MyISAM은 트랜잭션을 지원하지 않지만 InnoDB는 트랜잭션을 지원한다는 점이 큰 차이점

- MySQL 엔진 수준의 잠금에는 글로벌 잠금, 테이블 잠금, 네임드 잠금, 메타데이터 잠금 등 비교적 잠금 규모가 크고,
  스토리지 엔진 수준의 잠금에는 레코드 잠금, 간격 잠금, 다음 키 잠금, 자동 증가 잠금 등 비교적 잠금 규모가 작음
- MySQL 엔진 수준의 잠금은 모든 스토리지 엔진에 영향을 끼치지만, 스토리지 엔진 수준의 잠금은 해당 스토리지 엔진에만 적용

02

레코드 잠금

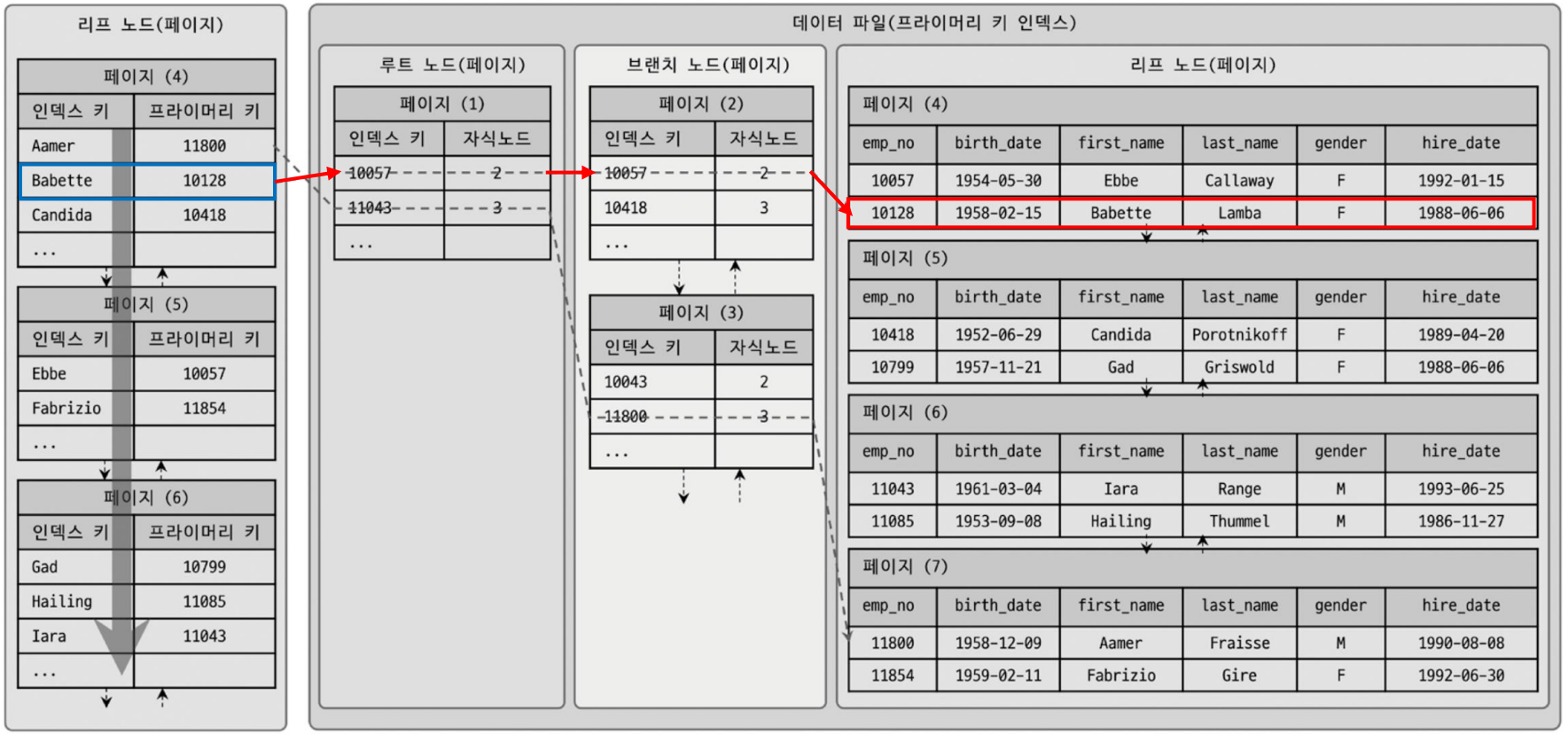
레코드 잠금 *(Record lock)*

- 일반적인 DBMS에서는 레코드 자체만을 잠그는 잠금
- InnoDB는 레코드를 잠그는 것이 아닌 인덱스의 레코드에 잠금을 걺


```
SELECT * FROM employees WHERE emp_no = 10128 FOR UPDATE;
```

레코드 잠금 *(Record lock)*

- 일반적인 DBMS에서는 레코드 자체만을 잠그는 잠금
- InnoDB는 레코드를 잠그는 것이 아닌 인덱스의 레코드에 잠금을 걺




```
SELECT * FROM employees WHERE first_name = 'Babette' FOR UPDATE;
```

레코드 잠금 *(Record lock)*

공유 잠금 *(Shared lock)*

- 여러 세션에서 동시에 레코드를 조회할 수 있으나 레코드 수정, 삭제는 할 수 없도록 하는 잠금
- SELECT … LOCK IN SHARE MODE 또는 SELECT … FOR SHARE 쿼리로 레코드를 명시적으로 잠글 수 있음
- 외래 키 제약 조건이 있는 테이블에 레코드 삽입 시 외래 키를 주요 키로 하는 테이블 레코드에 잠금을 걺

  이는 참조 무결성*을 보장하기 위해 공유 잠금을 걸어 삭제를 방지하는 것임

  * A를 참조하는 B 레코드를 삽입 중에 A 레코드가 삭제되는 것을 막기 위함

배타적 잠금 *(Exclusive lock)*

- 한 세션에서 배타적 잠금을 걸면 반환 전까지 다른 세션에서 공유 잠금이나 배타적 잠금을 할 수 없도록 하는 잠금
- 배타적 잠금이 걸린 레코드이더라도 다른 세션에서 동시에 레코드를 조회할 수는 있음
- SELECT … FROM UPDATE 쿼리로 레코드를 명시적으로 잠글 수 있음
- 재고가 하나인 상품에 여러 명의 사용자가 동시에 주문을 하는 상황에서 한 명의 사용자 외에는 모두 충돌 발생을 알게 되므로

  트랜잭션 롤백 등 복잡한 처리를 하지 않아도 됨

레코드 잠금 *(Record lock)*

확인하기

세션 1

```
1  START TRANSACTION;
2
3  SELECT * FROM employees WHERE emp_no = 10128
4      FOR SHARE;
5  -- 커밋 보류
```

세션 2

```
1  UPDATE employees SET hire_date = hire_date   6 s
2                   WHERE emp_no = 10128; -- 대기
```

잠금 상태

	trx_id	conn_id	tbl	idx	LOCK_TYPE	LOCK_MODE	LOCK_STATUS	LOCK_DATA
1	3090	10	employees	PRIMARY	RECORD	X,REC_NOT_GAP	WAITING	10128
2	3090	10	employees	<null>	TABLE	IX	GRANTED	<null>
3	413750404095192	9	employees	<null>	TABLE	IS	GRANTED	<null>
4	413750404095192	9	employees	PRIMARY	RECORD	S,REC_NOT_GAP	GRANTED	10128

레코드 잠금 *(Record lock)*

확인하기

세션 1

```
1  START TRANSACTION;
2
3  UPDATE employees SET hire_date = hire_date
4                   WHERE emp_no = 10128;
5  -- 커밋 보류
```

세션 2

```
1  UPDATE employees SET hire_date = hire_date   42 s
2                   WHERE emp_no = 10128; -- 대기
```

잠금 상태

	trx_id	conn_id	tbl	idx	LOCK_TYPE	LOCK_MODE	LOCK_STATUS	LOCK_DATA
1	2620	10	employees	*<null>*	TABLE	IX	GRANTED	*<null>*
2	2620	10	employees	PRIMARY	RECORD	X,REC_NOT_GAP	GRANTED	10128
3	2621	16	employees	PRIMARY	RECORD	X,REC_NOT_GAP	WAITING	10128
4	2621	16	employees	*<null>*	TABLE	IX	GRANTED	*<null>*

레코드 잠금 *(Record lock)*

확인하기

세션 1

```
1  START TRANSACTION;
2
3  UPDATE employees SET hire_date = hire_date
4                   WHERE first_name = 'Babette';
5  -- 커밋 보류
```

세션 2

```
1  UPDATE employees SET gender = gender  5 s
2                   WHERE emp_no = 10128; -- 대기
```

잠금 상태

	trx_id	conn_id	tbl	idx	LOCK_TYPE	LOCK_MODE	LOCK_STATUS	LOCK_DATA
1	2622	10	employees	*<null>*	TABLE	IX	GRANTED	*<null>*
2	2622	10	employees	idx_first_name	RECORD	X	GRANTED	'Babette', 10128
3	2622	10	employees	idx_first_name	RECORD	X,GAP	GRANTED	'Candida', 10418
4	2622	10	employees	PRIMARY	RECORD	X,REC_NOT_GAP	GRANTED	10128
5	2623	16	employees	PRIMARY	RECORD	X,REC_NOT_GAP	WAITING	10128
6	2623	16	employees	*<null>*	TABLE	IX	GRANTED	*<null>*

03

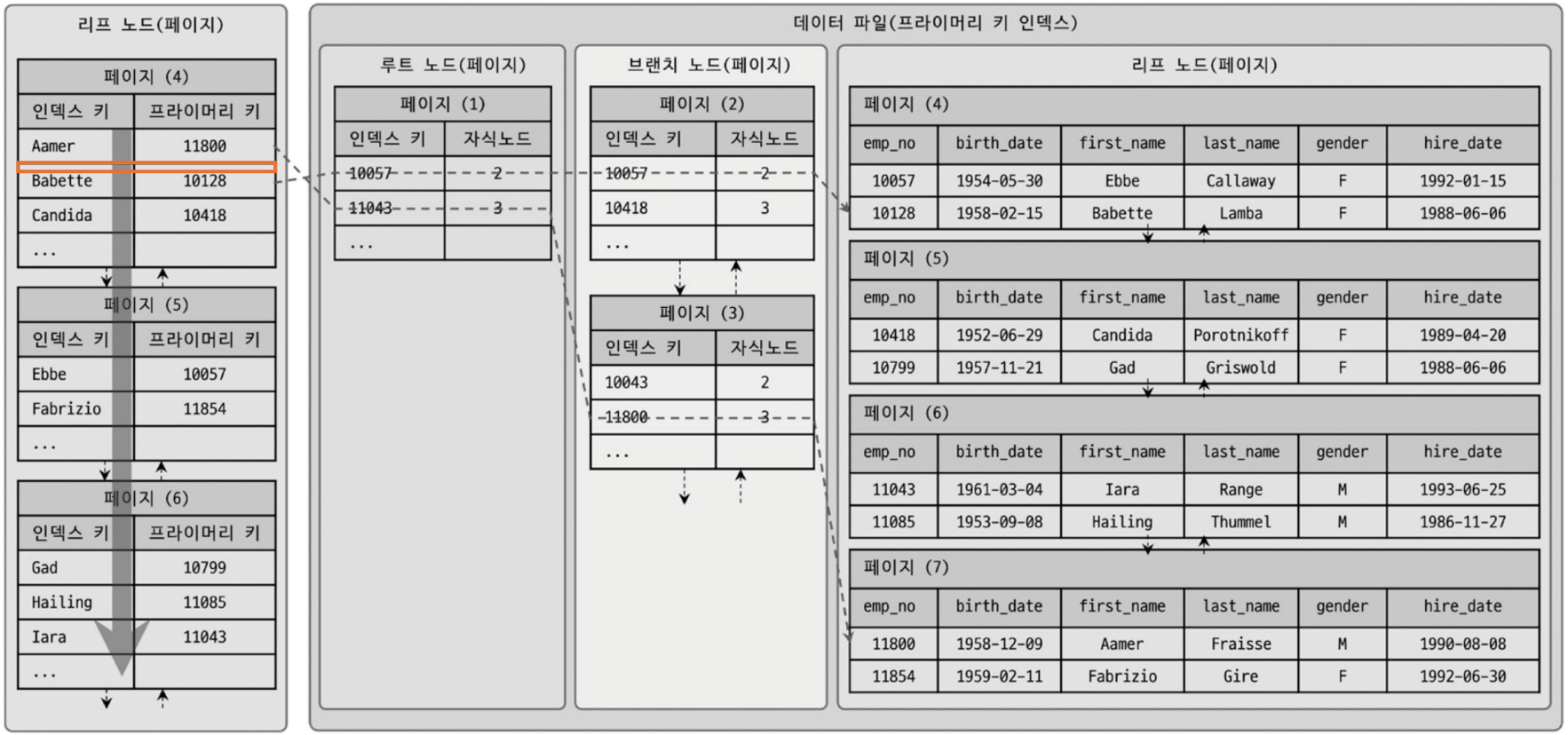
간격 잠금

간격 잠금 *(Gap lock)*

- 레코드 자체가 아니라 레코드와 바로 인접한 레코드 사이의 간격만 잠그는 잠금
- 레코드와 레코드 사이에 새로운 레코드가 생성되는 것을 제어함
- 외래 키 제약 조건이 있는 테이블에 레코드 삽입 시 외래 키를 주요 키로 하는 테이블에 해당 레코드가 없는 경우

  그 주요 키에 해당하는 인덱스 주위로 간격 잠금을 걸어 동시에 레코드가 삽입되는 것을 방지하여 삽입 시점에 레코드가 없음을 보장함
- 일반적으로 간격 잠금만 사용하기보다 간격 잠금과 레코드 잠금을 합친 다음 키 잠금을 많이 사용함

간격 잠금 *(Gap lock)*




```
SELECT * FROM employees WHERE first_name > 'Aamer' AND first_name < 'Babette' FOR UPDATE;
```

간격 잠금 *(Gap lock)*

확인하기

- 외래 키 제약 조건이 있는 테이블에 레코드 삽입 시 외래 키를 주요 키로 하는 테이블에 해당 레코드가 없는 경우

  그 주요 키에 해당하는 인덱스 주위로 간격 잠금을 걸어 동시에 레코드가 삽입되는 것을 방지하여 삽입 시점에 레코드가 없음을 보장함

세션 1

```
START TRANSACTION;
-- 세션 2에서 삽입 쿼리 발생
INSERT INTO teams (team_name, team_lead)
        VALUES ( team_name 'Development A',
                 team_lead 15000);
-- 커밋 보류
```

세션 2

```
INTO employees(emp_no,birth_date,first_name,last_name,
VALUES ( emp_no 150000,
         birth_date '1980-01-01',
         first_name 'John',
         last_name 'Doe',
         gender 'M',
         hire_date '2020-01-01');
```

잠금 상태

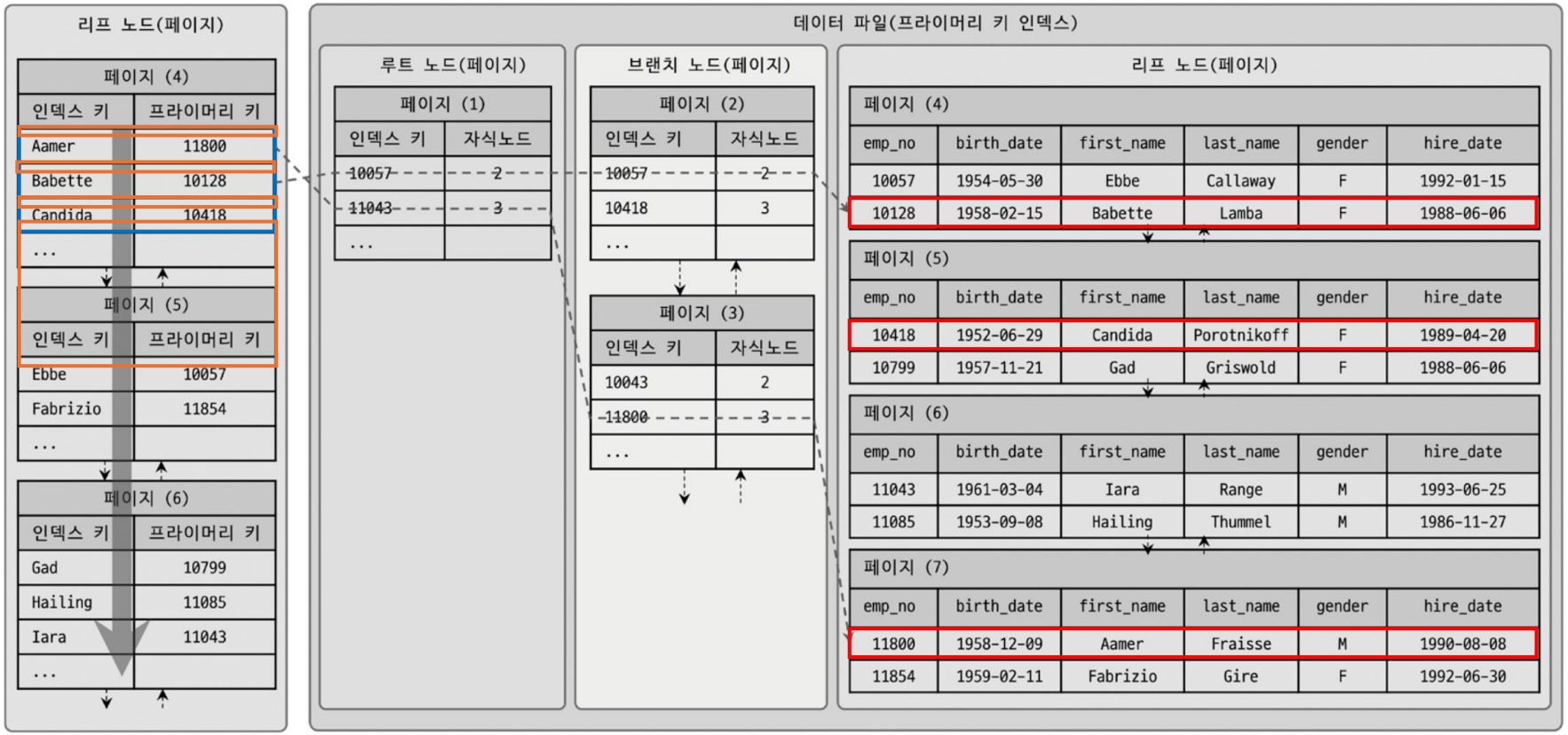
	trx_id	conn_id	tbl	idx	LOCK_TYPE	LOCK_MODE	LOCK_STATUS	LOCK_DATA
1	2637	10	teams	*<null>*	TABLE	IX	GRANTED	*<null>*
2	2637	10	employees	*<null>*	TABLE	IS	GRANTED	*<null>*
3	2637	10	employees	PRIMARY	RECORD	S,GAP	GRANTED	150000
4	2637	10	teams	PRIMARY	RECORD	X	GRANTED	supremum pseudo-record
5	2637	10	teams	team_name	RECORD	X	GRANTED	supremum pseudo-record

04

다음 키 잠금

다음 키 잠금 *(Next key lock)*

- 레코드 잠금과 간격 잠금을 합친 형태의 잠금




```
SELECT * FROM employees WHERE first_name >= 'Aamer' AND first_name < 'Ebbe' FOR UPDATE;
```

다음 키 잠금 *(Next key lock)*

확인하기

세션 1

```sql
START TRANSACTION;

UPDATE employees SET hire_date = hire_date
                 WHERE first_name ≥ 'Aamer'
                 AND first_name < 'Ebbe';
-- 커밋 보류
```

세션 2

```sql
INSERT INTO employees(emp_no,birth_date,first_
VALUES ( emp_no 12001,
        birth_date '1990-01-01',
        first_name 'Baker',
        last_name 'New',
        gender 'M',
        hire_date CURRENT_DATE);
```

잠금 상태

	trx_id	conn_id	tbl	idx	LOCK_TYPE	LOCK_MODE	LOCK_STATUS	LOCK_DATA
1	2633	10	employees	*<null>*	TABLE	IX	GRANTED	*<null>*
2	2633	10	employees	idx_first_name	RECORD	X	GRANTED	'Aamer', 11800
3	2633	10	employees	idx_first_name	RECORD	X	GRANTED	'Babette', 10128
4	2633	10	employees	idx_first_name	RECORD	X	GRANTED	'Candida', 10418
5	2633	10	employees	idx_first_name	RECORD	X	GRANTED	'Ebbe', 10057
6	2633	10	employees	PRIMARY	RECORD	X,REC_NOT_GAP	GRANTED	10057
7	2633	10	employees	PRIMARY	RECORD	X,REC_NOT_GAP	GRANTED	10128
8	2633	10	employees	PRIMARY	RECORD	X,REC_NOT_GAP	GRANTED	10418
9	2633	10	employees	PRIMARY	RECORD	X,REC_NOT_GAP	GRANTED	11800
10	2634	16	employees	idx_first_name	RECORD	X,GAP,INSERT_INTENTION	WAITING	'Candida', 10418
11	2634	16	employees	*<null>*	TABLE	IX	GRANTED	*<null>*

05

자동 증가 잠금

자동 증가 잠금 *(Auto increment lock)*

- AUTO_INCREMENT 속성이 적용된 컬럼의 값이 하나씩 증가됨을 보장하기 위해 사용하는 잠금
- InnoDB의 다른 잠금과 다르게 트랜잭션과 무관히 새로운 레코드를 저장하는 쿼리문 시작 시 획득해 끝날 때 해제됨
- MySQL 5.1 이상에서는 AUTO_INCREMENT 카운터에서 값을 얻는 구간에만 뮤텍스를 적용

들어 주셔서 감사합니다

E-mail	park@duck.com
LinkedIn	yeonjong-park
Instagram	yeonjong.park
GitHub	patulus

2025.08.20.

System Software Lab.