NHN FORWARD

천방지축 RDBMS 때려잡기
: Oracle to MySQL 변환기

다룰 내용

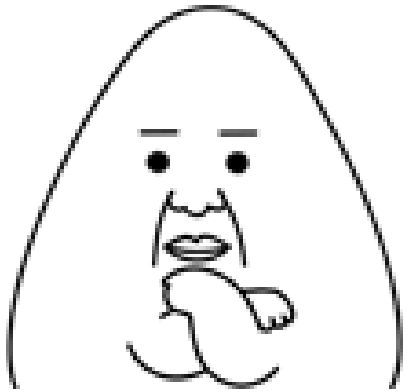
1. 우리가 이걸 왜 했을까
2. MySQL 변환 과정
3. 트러블슈팅
4. 당부의 말

우리가 이걸 왜 했을까

우리가 이걸 왜 했을까

제일 큰 이유는

- 비용 문제
- 노후 시스템 개선 의지
- 외부 ERP 사업



MySQL 변환 과정

MySQL 변환 과정

Oracle과 MySQL의 타입

구분	Oracle	MySQL
날짜	DATE	DATE, DATETIME
숫자	NUMBER	BIGINT, DECIMAL
대용량 텍스트	CLOB	TEXT, LONGTEXT

MySQL 변환 과정

subquery alias

- MySQL에서는 subquery에 alias가 없을 경우 아래와 같은 쿼리 에러가 발생

  Every derived table must have its own alias

MySQL 변환 과정

대문자와 소문자는 달라요

- Oracle은 select 시 컬럼, 테이블 명의 대소문자를 구분하지 않음(기본값)
- MySQL은 대소문자를 구분

MySQL 변환 과정

Oracle 함수 VS MySQL 함수

구분	Oracle	MySQL
NULL 처리	NVL	IFNULL
현재 날짜와 시간	SYSDATE	NOW
조건식	DECODE	CASE
문자 변환	TO_CHAR	CAST
날짜 계산	ADD_MONTHS	DATE_ADD

MySQL 변환 과정

Oracle 만의 독특한 구문/함수

- SEQUENCE
- ROWNUM
- MERGE INTO
- FULL OUTER JOIN
- PIVOT
- START WITH… CONNECT BY

MySQL 변환 과정

Oracle Sequence 이젠 안녕

- MySQL은 Sequence를 지원하지 않음
- 대신 AUTO_INCREMENT를 사용

MySQL 변환 과정

Oracle Sequence 이젠 안녕

<Oracle>

```
SELECT sequence_s.NEXTVAL FROM dual;
```

<MySQL>

```
INSERT sequence_s VALUES();
```

MySQL 변환 과정

Oracle Sequence 이젠 안녕

<Oracle>

```
SELECT sequence_s.NEXTVAL FROM dual; -- CURRVAL: 123456
```

<MySQL>

```
INSERT sequence_s VALUES();
ALTER TABLE sequence_s AUTO_INCREMENT = 123456;
```

MySQL 변환 과정

ROWNUM

- 쿼리 결괏값의 행의 개수를 제한할 수 있음
- 조회된 행이 몇 번째 행인지 알려줌

MySQL 변환 과정

ROWNUM

- 쿼리 결괏값의 수를 제한할 때는 LIMIT으로
- 조회된 행이 몇 번째 행인지는 ROW_NUMBER() OVER()로

MySQL 변환 과정

ROWNUM

<MySQL 5>

```
SELECT @rownum:= @rownum + 1 AS rn
  FROM table
     ,(SELECT @rownum:= 0) r
 WHERE id BETWEEN 1 AND 100
```

<MySQL 8>

```
SELECT ROW_NUMBER() OVER() AS rn
  FROM table
 WHERE id BETWEEN 1 AND 100
```

MySQL 변환 과정

Limit is zero-based Index

- ROWNUM은 1-based Index
- Limit은 0-based Index

Oracle	MySQL
ROWNUM 5 between 10	LIMIT 4, 5

MySQL 변환 과정

Limit is zero-based Index

<Oracle>

```
SELECT *
  FROM
     (
      SELECT a.id
           , b.name
        FROM table_a a
           , table_b b
       WHERE a.id = b.id
       ORDER BY a.id
       )
 WHERE ROWNUM <= 5
```

MySQL 변환 과정

Limit is zero-based Index

<Oracle>

```
SELECT *
  FROM
     (
      SELECT a.id
           , b.name
        FROM table_a a
           , table_b b
       WHERE a.id = b.id
       ORDER BY a.id
       )
 WHERE ROWNUM <= 5
```

<MySQL>

```
SELECT a.id
     , b.name
  FROM table_a a
     , table_b b
 WHERE a.id = b.id
 ORDER BY a.id
 LIMIT 5
```

MySQL 변환 과정

(+) Outer Join

- Oracle에서 지원하는 독특한 문법(+)
- ANSI SQL로 변경

<Oracle>

```
SELECT a.id
     , b.name
  FROM table_a a
     , table_b b
 WHERE a.id = b.id(+)
```

<MySQL>

```
SELECT a.id
     , b.name
  FROM table_a a
       LEFT OUTER JOIN table_b b
         ON a.id = b.id
```

MySQL 변환 과정

FULL OUTER JOIN

<Oracle>

```
SELECT a.id
     , b.name
  FROM table_a a
     , table_b b
 WHERE a.id(+) = b.id(+)
```

MySQL 변환 과정

FULL OUTER JOIN

<Oracle>

```
SELECT a.id
     , b.name
  FROM table_a a
     , table_b b
 WHERE a.id(+) = b.id(+)
```

<MySQL>

```
SELECT a.id
     , b.name
  FROM table_a a
       LEFT OUTER JOIN table_b b
       ON a.id = b.id
UNION
SELECT a.id
     , b.name
  FROM table_b b
       LEFT OUTER JOIN table_a a
       ON b.id = a.id
```

MySQL 변환 과정

MERGE INTO는

- INSERT ON DUPLICATE KEY UPDATE로

<Oracle>

```
MERGE
 INTO table_a a
USING table_b b
   ON(a.id = b.id)
 WHEN MATCHED THEN -- 일치
      UPDATE SET a.name = b.name
 WHEN NOT MATCHED THEN -- 불일치
      INSERT(id, name)
      VALUES(1, '홍길동')
```

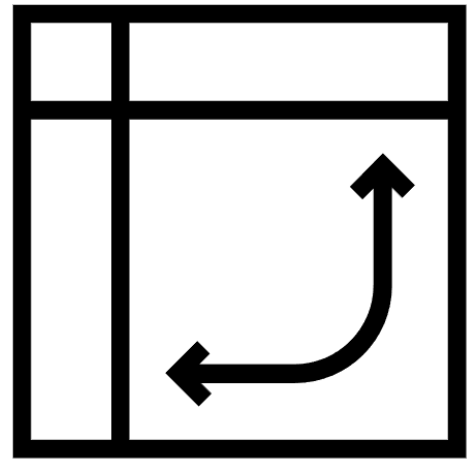
<MySQL>

```
INSERT
  INTO table_a
    (id, name)
 VALUES
    (1, '홍길동')
 ON DUPLICATE Key UPDATE name = '홍길동'
```

MySQL 변환 과정

PIVOT

- Oracle에서 행과 열을 변환할 때 사용
- PIVOT 대신 DECODE로 작성하는 방법도 있음



MySQL 변환 과정

PIVOT

<Oracle>

```
SELECT *
  FROM(
        SELECT name
             , salary
             , month
          FROM emp_salary
      )
 PIVOT(
        SUM(NVL(salary, 0))
        FOR month IN
              (1, 2, 3, 4, 5, 6)
      )
```

MySQL 변환 과정

PIVOT

<Oracle>

```
SELECT *
  FROM(
        SELECT name
             , salary
             , month
          FROM emp_salary
      )
 PIVOT(
        SUM(NVL(salary, 0))
        FOR month IN
              (1, 2, 3, 4, 5, 6)
      )
```

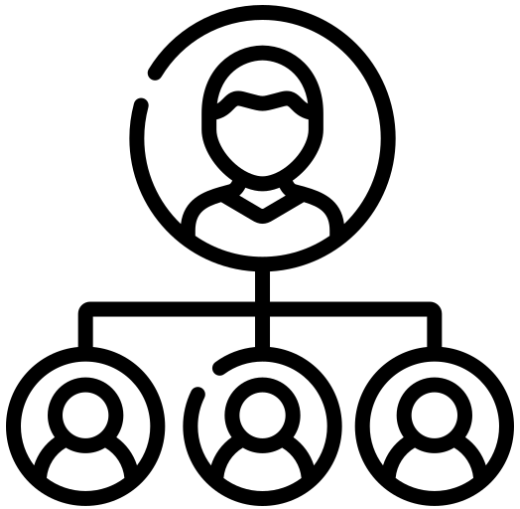
<MySQL>

```
SELECT name
 , SUM(IF(month = 1, IFNULL(salary, 0), 0) AS `1`
 , SUM(IF(month = 2, IFNULL(salary, 0), 0) AS `2`
 , SUM(IF(month = 3, IFNULL(salary, 0), 0) AS `3`
 , SUM(IF(month = 4, IFNULL(salary, 0), 0) AS `4`
 , SUM(IF(month = 5, IFNULL(salary, 0), 0) AS `5`
 , SUM(IF(month = 6, IFNULL(salary, 0), 0) AS `6`
  FROM emp_salary
 GROUP BY name
```

MySQL 변환 과정

계층형 쿼리(Tree)

- START WITH … CONNECT BY



MySQL 변환 과정

계층형 쿼리(Tree)

```
WITH RECURSIVE cte(parent_id, id) AS
(
  /* 재귀 시작 데이터, START WITH에 해당 */
  SELECT a.parent_id, a.id
    FROM table a
   WHERE a.parent_id IS NULL /* 시작조건 */
   UNION ALL /* 중복제거를 원한다면 UNION 써도 됨 */
  /* 재귀 조건 CONNECT BY에 해당 */
  SELECT a.parent_id, a.id
    FROM table a
   INNER JOIN cte ON cte.id = a.parent_id /* 재귀조건 */
)
SELECT *
  FROM cte
```

트러블슈팅

다르게 동작해요

Q.

```
SELECT ‘A’ || ‘B’ FROM dual;
```

다르게 동작해요

Q.

```
SELECT 'A' || 'B' FROM dual;
```

A.

Oracle	MySQL
AB	0(FALSE)

다르게 동작해요

||(PIPES_AS_CONCAT)

```
SELECT 'A' || 'B' FROM dual;
```

Oracle	MySQL
AB	AB

다르게 동작해요

||(PIPES_AS_CONCAT)

```
SELECT 'A' || NULL || 'B' FROM dual;
```

Oracle	MySQL
AB	NULL

다르게 동작해요

||(PIPES_AS_CONCAT)

```
SELECT 'A' || NULL || 'B' FROM dual;
```

CONCAT_WS

```
SELECT CONCAT_WS('', 'A', NULL, 'B') FROM dual;
```

Oracle	MySQL
AB	AB

다르게 동작해요

Q.

```
SELECT CASE WHEN '' IS NULL THEN 1 ELSE 2 END FROM dual;
```

다르게 동작해요

Q.

```
SELECT CASE WHEN '' IS NULL THEN 1 ELSE 2 END FROM dual;
```

A.

Oracle	MySQL
1	2

다르게 동작해요

‘’ != NULL

```
INSERT INTO table VALUES(‘’);
```

Oracle	MySQL
NULL	‘’

다르게 동작해요

‘ ’ != NULL

```
INSERT INTO table VALUES(NULLIF(NULL, ‘’));
```

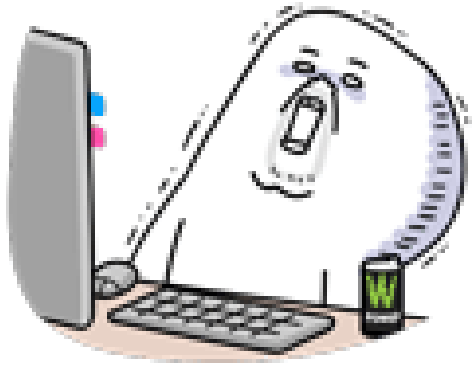
Oracle	MySQL
NULL	NULL

다르게 동작해요

DELETE의 악몽(feat. 데이터가 전부 어디 갔지)

- DELETE FROM table;

```
DELETE FROM table
<where>
  <if>
     ...
  </if>
</where>
```

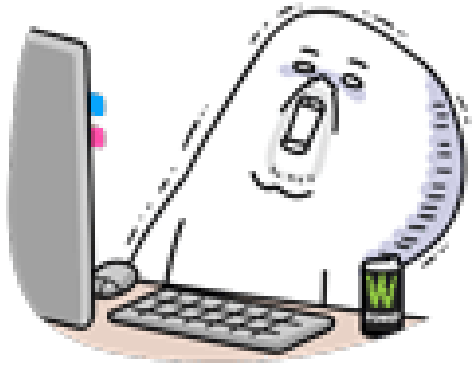


다르게 동작해요

DELETE의 악몽(feat. 데이터가 전부 어디 갔지)

- DELETE FROM table
  WHERE ;

```
DELETE FROM table
WHERE
<dynamic>
  <if>
     ...
  </if>
</dynamic>
```



다르게 동작해요

Byte

- Byte 입력
  - Oracle은 UTF-8 문자열에서 한글 입력 시 3byte, 영어/숫자는 1byte

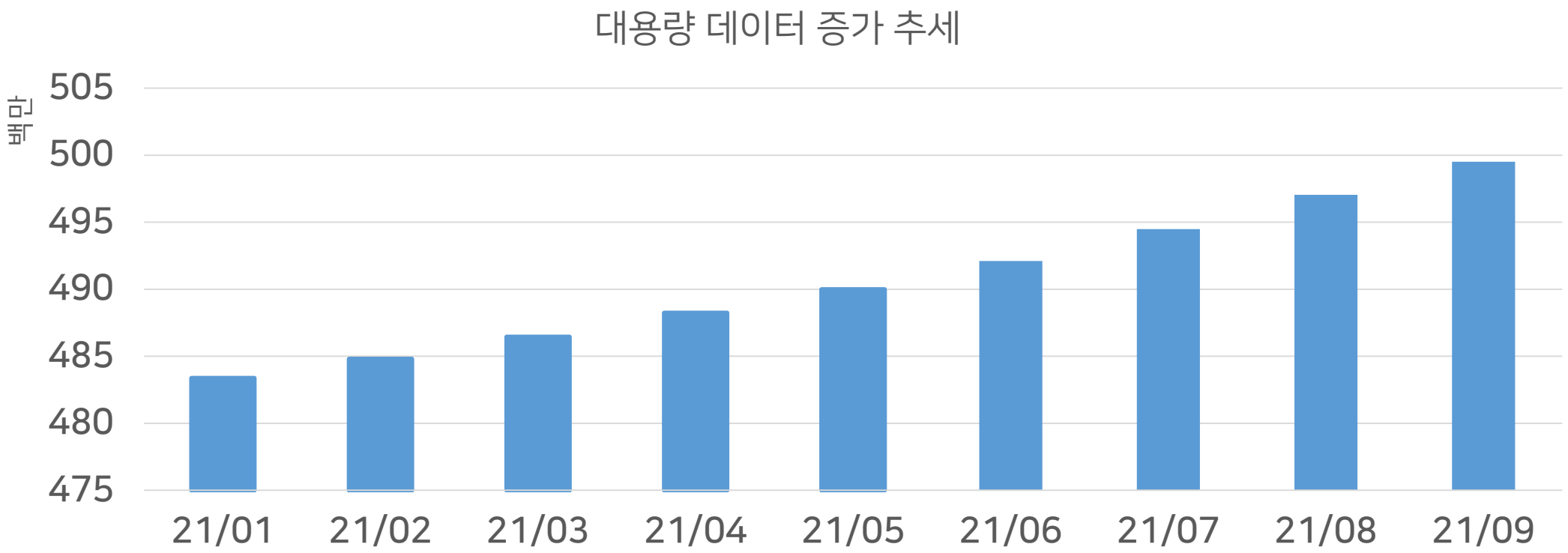
```
<Oracle>
INSERT table VALUES('안녕123ABC'); -- varchar(10)
/* ORA-01704 문자열이 너무 깁니다 */
```

- Byte 연산
  - SUBSTRB

대용량 시스템을 다루는 우리의 자세

너무 느려요

- 대용량 데이터와 복잡한 쿼리는 재무시스템의 숙명
- Oracle에서는 잘 동작하던 쿼리들이 MySQL에서 성능 이슈가 발생



대용량 시스템을 다루는 우리의 자세

너무 느려요

- 대용량 테이블 파티셔닝 진행

Partition Name	Position	테이블	Method	Expression	Description	Table Rows	Avg Row Len	Data Len	Max Data Len	Index Len
p202007	188	je_lines	RANGE COLUMNS	`JE_DATE`	'2020-08-01'	1,231,026	562	692,060,160	0	1,122,156,544
p202008	189	je_lines	RANGE COLUMNS	`JE_DATE`	'2020-09-01'	1,097,201	592	650,117,120	0	1,044,447,232
p202009	190	je_lines	RANGE COLUMNS	`JE_DATE`	'2020-10-01'	1,301,736	506	659,554,304	0	1,025,048,576
p202010	191	je_lines	RANGE COLUMNS	`JE_DATE`	'2020-11-01'	1,210,812	593	718,225,408	0	1,132,265,472
p202011	192	je_lines	RANGE COLUMNS	`JE_DATE`	'2020-12-01'	1,120,890	524	588,201,984	0	1,040,711,680
p202012	193	je_lines	RANGE COLUMNS	`JE_DATE`	'2021-01-01'	1,370,605	418	573,521,920	0	1,076,215,808
p202101	194	je_lines	RANGE COLUMNS	`JE_DATE`	'2021-02-01'	1,210,809	419	507,461,632	0	939,409,408
p202102	195	je_lines	RANGE COLUMNS	`JE_DATE`	'2021-03-01'	1,397,119	420	587,202,560	0	1,085,456,384
p202103	196	je_lines	RANGE COLUMNS	`JE_DATE`	'2021-04-01'	1,685,722	408	688,832,512	0	1,255,636,992
p202104	197	je_lines	RANGE COLUMNS	`JE_DATE`	'2021-05-01'	1,544,056	435	672,120,832	0	1,194,852,352
p202105	198	je_lines	RANGE COLUMNS	`JE_DATE`	'2021-06-01'	1,579,166	415	656,408,576	0	1,159,168,000
p202106	199	je_lines	RANGE COLUMNS	`JE_DATE`	'2021-07-01'	1,889,082	419	791,642,112	0	1,379,614,720
p202107	200	je_lines	RANGE COLUMNS	`JE_DATE`	'2021-08-01'	2,143,725	444	952,074,240	0	1,655,832,576
p202108	201	je_lines	RANGE COLUMNS	`JE_DATE`	'2021-09-01'	2,358,570	436	1,028,620,288	0	1,802,747,904

대용량 시스템을 다루는 우리의 자세

너무 느려요

<파티셔닝 제외 조건>

```
SELECT COUNT(*)
  FROM 테이블
 WHERE ID = 1 AND 생성일자 BETWEEN '2021-09-01' AND '2021-09-02'
(execution: 18 s 880 ms, fetching: 16 ms)
```

<파티셔닝 조건 포함>

```
SELECT COUNT(*)
  FROM 테이블
 WHERE ID = 1 AND 생성일자 BETWEEN '2021-09-01' AND '2021-09-02'
   AND 파티션조건 BETWEEN '2021-09-01' AND '2021-09-02'
(execution: 47 ms, fetching: 29 ms) -- 약 382배 성능 향상
```

대용량 시스템을 다루는 우리의 자세

너무 느려요

```
SELECT a.name
  FROM table a USE INDEX(IDX_N1)
 INNER JOIN table b ON a.id = b.id
 WHERE a.name = 'A'
```

대용량 시스템을 다루는 우리의 자세

너무 느려요

<Oracle>

```
SELECT a.name, b.age
  FROM table a
 INNER JOIN table b ON a.id = b.id
 WHERE a.name = ‘A’ OR b.age = 20
```

대용량 시스템을 다루는 우리의 자세

너무 느려요

<Oracle>

```
SELECT a.name, b.age
  FROM table a
 INNER JOIN table b ON a.id = b.id
 WHERE a.name = 'A' OR b.age = 20
```

<MySQL>

```
SELECT a.name, b.name
  FROM table a
 INNER JOIN table b ON a.id = b.id
 WHERE a.name = 'A'
 UNION ALL
SELECT a.name, b.name
  FROM table a
 INNER JOIN table b ON a.id = b.id
 WHERE b.name = 'B'
```

대용량 시스템을 다루는 우리의 자세

너무 느려요

- TEMPTABLE 방식의 View 사용은 이제 그만

Merge	Temptable
원본 테이블에 쿼리를 재작성	임시 테이블에 저장

대용량 시스템을 다루는 우리의 자세

너무 느려요

- TEMPTABLE 방식의 View 사용은 이제 그만
  - DISTINCT, GROUP BY
  - UNION
  - 서브쿼리
  - LIMIT
  - etc.

대용량 시스템을 다루는 우리의 자세

너무 느려요

- TEMPTABLE 방식의 View 사용은 이제 그만

```
<TEMPTABLE VIEW>
SELECT *
  FROM(
     SELECT MAX(a.id) AS id
          , a.name
       FROM table a
      GROUP BY a.name
       ) view
 WHERE view.name = 'A'
   AND view.id = 1
```

대용량 시스템을 다루는 우리의 자세

너무 느려요

- TEMPTABLE 방식의 View 사용은 이제 그만

<TEMPTABLE VIEW>

```
SELECT *
  FROM(
     SELECT MAX(a.id) AS id
          , a.name
       FROM table a
      GROUP BY a.name
       ) view
 WHERE view.name = 'A'
   AND view.id = 1
```

<VIEW 해체>

```
SELECT MAX(a.id) AS id
     , a.name
  FROM table a
 WHERE a.name = 'A'
   AND a.id = 1
 GROUP BY a.name
```

Isolation

Isolation Level은

Isolation

Isolation Level 종류

- READ UNCOMMITTED
  - COMMIT 되지 않은 데이터도 읽을 수 있도록 허용
- READ COMMITTED(Oracle의 기본 설정 값)
  - COMMIT된 데이터만 읽을 수 있을 수 있도록 허용
  - NON-REPEATABLE READ(조회할 때마다 다른 값이 조회됨)
- REPEATABLE READ(MySQL의 기본 설정 값)
  - 트랜잭션이 열리기 전에 COMMIT된 내용으로만 조회
  - UNDO 영역을 지속적으로 사용하여 성능 저하
- SERIALIZABLE
  - 조회 시에도 Lock을 거는 가장 엄격한 격리 Level

Isolation

Isolation Level 종류

- READ UNCOMMITTED
  - COMMIT 되지 않은 데이터도 읽을 수 있도록 허용

- READ COMMITTED(Oracle의 기본 설정 값)
  - COMMIT된 데이터만 읽을 수 있을 수 있도록 허용
  - NON-REPEATABLE READ(조회할 때마다 다른 값이 조회됨)

- REPEATABLE READ(MySQL의 기본 설정 값)
  - 트랜잭션이 열리기 전에 COMMIT된 내용으로만 조회
  - UNDO 영역을 지속적으로 사용하여 성능 저하

- SERIALIZABLE
  - 조회 시에도 Lock을 거는 가장 엄격한 격리 Level

Isolation

Isolation Level 종류

- REPEATABLE READ(MySQL의 기본 설정 값) 사용 이슈
  - 대부분의 로직이 READ_COMMITTED에 맞게 개발됨
  - 한 트랜잭션 안의 로직이 상당히 길었음
  - Lock 발생

Isolation

Isolation Level 종류

✔READ COMMITTED로 설정 변경

- 이미 운영 중인 시스템을 REPEATABLE READ에 맞게 바꾸기는 어렵다는 판단
- 최종적으로 READ COMMITTED로 설정 변경
- RDBMS 변환 시 Isolation Level 고민 필요

당부의 말

당부의 말

RDBMS 변환을 고려 중이라면

- 사전 준비 필수
- 충분한 시간을 가지고 접근
- 언제든 복구 가능한 형태로 데이터를 보관
- QA

당부의 말

RDBMS 변환을 고려 중이라면

- 사전 준비 필수
- 충분한 시간을 가지고 접근
- 언제든 복구 가능한 형태로 데이터를 보관
- QA

당부의 말

RDBMS 변환을 고려 중이라면

- 사전 준비 필수
- 충분한 시간을 가지고 접근
- 언제든 복구 가능한 형태로 데이터를 보관

QA

당부의 말

RDBMS 변환을 고려 중이라면

- 사전 준비 필수
- 충분한 시간을 가지고 접근
- 언제든 복구 가능한 형태로 데이터를 보관

.QA

NHN FORWARD