데이터베이스(다반)

20192875

조성현

중간고사 대체과제

**1. 트랜잭션의 4대 특징**

1. 원자성(Atomicity)

- 트랜잭션의 작업이 부분적으로 실행되거나 중단되지 않는 것

- 하나의 덩어리로 전부 되는 것 아니면 아무것도 되지 않음.

- 작업의 단위를 일부분만 실행하지 않음

1. 일관성(Consistency)

- 트랜잭션이 성공적으로 완료되면 일관적인 DB상태를 유지하는 것

ex) employees 테이블의 first\_name의 데이터 타입이 Char에서 int형으로 바뀌지 않음

- 트랜잭션이 진행되는 동안에 DB가 변경되더라도 업데이트된 DB로 트랜잭션

이 되는 것이 아니라 처음에 트랜잭션을 진행하기 위해 참조한 DB로 진행됨.

1. 격리성(Isolation)

- 트랜잭션 수행 시 다른 트랜잭션의 작업이 끼어들지 못하는 것

- 트랜잭션끼리 서로 간섭할 수 없음

1. 지속성(Durability)

- 성공적으로 수행된 트랜잭션은 영원히 반영이 되는 것

- 한번 commit을 하면 현재 상태는 영원히 보장됨

DB에서 일관성의 특징으로 사용자가 DB로 검색을 하는 등 이용할 때 서로 다른 사용자들이 일관성 있는 DB를 볼 수 있어서 서로 다른 DB를 보는 경우가 없다. 또한 일관성의 특징으로 employees 테이블에서 특정 사람이 부서를 옮겼을 때 개발 부서 테이블에서는 마케팅 부서로 옮겼다고 뜨는데 마케팅 부서에서는 아무 들어온 사람이 없는 경우가 없어야 한다. 일관성 특징으로 은행 DB에서 입출금 오류 등 심각한 문제를 막을 수 있어서 중요한 특징이다.

격리성은 한 DB에서 병행처리를 막아 공통된 데이터가 혼란스러워지는 경우를 막는 특징이다. 이와 같이 간섭이 일어날 경우 갱신분실, 오손판독, 반복불가능, 팬텀문제 등 여러가지 문제점들이 발생하기 때문에 Lock과 unlock 기능으로 격리성을 보장한다. 이 격리성은 DB를 이용하는데 서로 같은 데이터를 봐야하는 일관성 특징으로 비슷한 이유로 주요한 특징이다.

**2. MySQL의 인덱스 종류**

인덱스의 데이터 저장 방식별로 구분하는 것은 많은 분류가 가능하겠지만, 대표적으로 BTREE 인덱스와 HASH인덱스로 구분할 수 있다.

- BTREE

가장 일반적으로 사용되는 인덱스 알고리즘으로서, 칼럼의 값을 변형하지 않고, 원래의 값을 이용해 인덱싱하는 알고리즘

- HASH

컬럼의 값으로 해시 값을 계산해서 인덱싱하는 알고리즘으로, 매우 빠른 검색을 지원한다. 하지만 전방일치와 같이 값의 일부만 검색할 수 없고 비교문을 쓸 수 없다.

여기서 스토리지엔진에 따라 HASH를 못 쓸 수 있다.

MySQL 문법에서 테이블 생성 시 지정할 수 있는 인덱스 타입이 2개로 나타난다.

|  |
| --- |
| index\_type:  USING {BTREE | HASH} |

이는 문법 호환성을 위해서 문법적으로 허용만 할 뿐, MySQL은 스토리지엔진에 따라 지원되는 인덱스의 타입이 다르다.

|  |  |  |  |
| --- | --- | --- | --- |
| 스토리지엔진 | 지원되는 인덱스 | Full-text search indexes | Clustered 인덱스 |
| InnoDB | BTREE | No | Yes |
| Memorry/HEAP | HASH, BTREE | No | No |
| NDB | HASH, BTREE | No | No |
| MyISAM | BTREE, RTREE | Yes | No |
| Archive |  | No | No |

Clustered Index

테이블 레코드들이, 인덱스 컬럼의 정렬 순서대로 적재되어 있는 것. 인덱스 자체가 테이블 레코드 순서와 같다. 인덱스 자체의 리프 페이지가 곧 데이터이다. 그러므로 인덱스 자체에 데이터가 포함되어 있다고 볼 수 있다.

MySQL에서 InnoDB를 스토리지엔진 기반에서의 경우

테이블에서 Primary key가 있다면 Primary key를 Clustered 인덱스로, 없다면 UNIQUE 하면서 NOT NULL인 컬럼을 Clustered 인덱스로, 그것도 없다면 임의로 보이지 않는 컬럼을 만들어 Clustered 인덱스로 만든다.

Secondary Index

보조 인덱스의 생성시에는 데이터 페이지는 그냥 둔 상태에서 별도의 페이지에 인덱스를 구성한다. 보조 인덱스의 인덱스 자체의 리프 페이지는 데이터가 아니라 데이터가 위치하는 주소값을 가진다.

Clustered 인덱스는 Secondary 인덱스 보다 검색(R) 속도는 더 빠르다. 하지만 입력/수정/삭제(CUD)는 느리다.

**3. 와일드카드(like)기반 검색**

검색하려는 값을 정확하게 모를 경우에도 검색할 수 있도록 하는 것

|  |  |
| --- | --- |
| 와일드 카드 | 기술 |
| % | 0개 이상의 문자를 대체 |
| \_ | 단일 문자에 대한 대체 |
| [charlist] | 문자 세트 및 범위가 일치 |
| [^charlist] 또는 [!charlist] | 문자 세트를 제외한 문자들로 대체 |

-예시

1) A%

‘A’로 시작하는 모든 문자열

2) ‘%A’%

‘A’가 포함된 모든 문자열

3) ‘\_A%’

두 번째 문자가 ‘A’인 모든 문자열

4) ‘[ABC]%’

첫 번째 문자가 ‘A’ 또는 ‘B’ 또는 ‘C’인 모든 문자열

5) ‘[^A]%’

첫 번째 문자가 ‘A’가 아닌 모든 문자열

6)[0-9][0-9][0-9][0-9]

4자리 숫자를 가지는 문자열

-단점

와일드카드 문자를 이용한 검색은 텍스트 열(문자열)에서만 사용할 수 있다.

NULL값은 와일드 카드로 매칭되지 않음

‘%’나 ‘\_’ 또는 ‘[‘ 등 와일드카드 연산자를 검색할 때는 주의가 필요함

**4. 오라클의 데이터 타입과 MySQL의 데이터타입**

***- Oracle 데이터 타입***

A. 문자형 데이터 타입

|  |  |
| --- | --- |
| 데이터 유형 | 정의 |
| CHAR(n) | 고정 길이 데이터 타입(최대 2000byte)- 지정된 길이보다 짧은 데이터 입력 시 나머지 공간을 공백으로 채움 |
| VARCHAR2(n) | 가변 길이 데이터 타입(최대 4000byte)- 지정된 길이보다 짧은 데이터 입력 시 나머지 공간은 채우지 않음 |
| NCHAR(n) | 고정 길이 유니코드 데이터 타입(최대 2000 byte) |
| NBARCHAR2(n) | 가변 길이 데이터 타입(최대 4000byte) |
| LONG | 가변 길이 데이터 타입(최대 2Gbyte) |
| CLOB | 대용량 텍스트 데이터 타입(최대 4Gbyte) |
| NCLOB | 대용량 텍스트 유니코드 데이터 타입(최대 4Gbyte) |

B. 숫자형 데이터 타입

|  |  |
| --- | --- |
| 데이터 유형 | 정의 |
| BINARY\_FLOAT | 부동 소수형 데이터 타입(4byte)- 32bit 부동 소수 |
| BINARY\_DOUBLE | 부동 소수형 데이터 타입(8byte)- 64bit 부동 소수 |
| NUMBER(P, S) | P, S로 표현 숫자 데이터 타입 p: 1~38, s: 84~127 p(precision): 유효 자리수, s(scale): 수소점 유효자리 |

C. 날짜형 데이터 타입

|  |  |
| --- | --- |
| 데이터 유형 | 정의 |
| DATE | 고정 길이 날짜 |
| INTERVAL\_YEAR | 날짜(년도, 월) 형태의 기간 표현 데이터 타입 |
| INTERVAL\_DAY | 날짜 및 시간(요일, 시, 분, 초)형태의 기간 표현 데이터 타입 |
| TIMESTAMP | 밀리초(ms)까지 표현 데이터 타입 |
| TIMESTAMP\_WITH TIME ZONE | 날짜 및 시간대 형태의 데이터 타입 |
| TIMESTAMP\_WITH LOCAL TIME ZONE | 저장 시 데이터베이스 시간대를 준수, 조회 시 조회하는 클라이언트 시간 표현 데이터 타입 |

D. 이진 데이터 타입

|  |  |
| --- | --- |
| 데이터 유형 | 정의 |
| RAW(n) | 가변 길이 이진 데이터 타입(최대 2Gbyte) |
| LONGRAW | 가변 길이 이진 데이터 타입(최대 4Gbyte) |
| BLOB | 대용량의 바이너리 데이터를 저장하기 위한 데이터 타입(최대 4Gbyte) |
| BFILE | 대용량의 바이너리 데이터를 파일형태로 저장하기 위한 데이터 타입(최대 4Gbyte) |

***-MySQL 데이터 타입***

A. 문자형 데이터 타입

|  |  |
| --- | --- |
| 데이터 유형 | 정의 |
| CHAR(n) | 고정 길이 데이터 타입(최대 255byte)- 지정된 길이보다 짧은 데이터 입력 시 나머지 공간을 공백으로 채움 |
| VARCHAR(n) | 가변 길이 데이터 타입(최대 65535byte)- 지정된 길이보다 짧은 데이터 입력 시 나머지 공간을 채우지 않음 |
| TINYTEXT(n) | 문자열 데이터 타입(최대 255byte) |
| TEXT(n) | 문자열 데이터 타입(최대 65535byte) |
| MEDIUMTEXT(n) | 문자열 데이터 타입(최대 16777215byte) |
| LONGTEXT(n) | 문자열 데이터 타입(최대 4294967295byte) |

B. 숫자형 데이터 타입

|  |  |
| --- | --- |
| 데이터 유형 | 정의 |
| TINYINT(n) | 정수형 데이터 타입(1byte) -128~+127 또는 0~255수로 표현 가능함 |
| SMALLINT(n) | 정수형 데이터 타입(2byte) -32768~+32767 또는 0~65536수로 표현 가능함 |
| MEDIUMINT(n) | 정수형 데이터 타입(3byte) -8388608~+8388607 또는 0~16777215수로 표현 가능함 |
| INT(n) | 정수형 데이터 타입(4byte) -2147483648~+2147483647 또는 0~4294967295수로 표현 가능함 |
| BIGINT(n) | 정수형 데이터 타입(8byte) 무제한 수로 표현 가능함 |
| FLOATINT(길이, 소수) | 부동 소수형 데이터 타입(4byte) 고정 소수점을 사용한 형태임 |
| DECIMAL(길이, 소수) | 고정소수형 데이터 타입고정(길이 +1byte) 소수점을 사용한 형태임 |
| DOUBLE(길이, 소수\_ | 부동 소수형 데이터 타입(8byte) -DOUBLE을 문자열로 저장함 |

C. 날짜형 데이터 타입

|  |  |
| --- | --- |
| 데이터 유형 | 정의 |
| DATE | 날짜(년도, 월, 일)형태의 기간 표현 데이터 타입(3byte) |
| TIME | 시간(시, 분, 초)형태의 기간 표현 데이터 타입(3byte) |
| DATETIME | 날짜와 시간 형태의 기간 표현 데이터 타입(8byte) |
| TIMESTAMP | 날짜와 시간 형태의 기간 표현 데이터 타입(4byte) – 시스템 변경 시 자동으로 그 날짜와 시간이 저장됨 |
| YEAR | 년도 표현 데이터 타입(1byte) |

D. 이진 데이터 타입

|  |  |
| --- | --- |
| 데이터 유형 | 정의 |
| BINARY(n) & BYTE(n) | CHAR형태의 이진 데이터 타입(최대 255byte) |
| VARBINARY(n) | VARCHAR형태의 이진 데이터 타입(최대 65535byte) |
| TINYBLOB(n) | 이진 데이터 타입(최대 255byte) |
| BLOB(n) | 이진 데이터 타입(최대 16777245byte) |
| LONGBLOB(n) | 이진 데이터 타입(최대 4294967295byte) |

- Oracle과 MySQL의 데이터 타입 차이

|  |  |  |
| --- | --- | --- |
| MySQL | Size | Oracle |
| BIGINT | 8 Bytes | NUMBER (19,0) |
| BIT | approximately (M+7)/8 Bytes | RAW |
| DECIMAL(M,D) | M+2 bytes if D > 0, M+1 bytes if D = 0 (D+2, if M < D) | FLOAT(24), BINARY\_FLOAT |
| DOUBLE | 8 Bytes | FLOAT(24), BINARY\_FLOAT, BINARY\_DOUBLE |
| DOUBLE PRECION | 8 Bytes | FLOAT(24), BINARY\_DOUBLE |
| FLOAT(25<=X <=53) | 8 Bytes | FLOAT(24), BINARY\_FLOAT |
| FLOAT(X<=24) | 4 Bytes | FLOAT, BINARY\_FLOAT |
| INT | 4 Bytes | NUMBER (10,0) |
| INTEGER | 4 Bytes | NUMBER (10,0) |
| MEDIUMINT | 3 Bytes | NUMBER (7,0) |
| NUMERIC | M+2 bytes if D > 0, M+1 bytes if D = 0 (D+2, if M < D) | NUMBER |
| REAL | 8 Bytes | FLOAT(24), BINARY\_FLOAT |
| SMALLINT | 2 Bytes | NUMBER(5,0) |
| TINYINT | 1 Byte | NUMBER(3,0) |

**5. Oracle 조인 타입**

- Join 종류

1. EQUIJOIN

column 간의 값들이 서로 정확히 일치하는 경우에 사용할 수 있음. PK=FK

2. NON – EQUIJOIN

두 개의 테이블 간에 컬럼 값들이 서로 정확하게 일치하지 않는 경우 사용함

3. OUTER JOIN

두 개의 테이블 간의 JOIN 걸었을 때, JOIN 조건 만족하지 않은 경우에도 +연산자를 사용하여 조인할 수 있음

left와 right로 나뉨

4. SELF JOIN

동일한 테이블에 대한 조인할 때 사용

-표준조인(ANSI Join)과 차이점

Oracle에서도 표준조인 문법을 문제없이 사용할 수 있다.

표준조인은 조인의 종류를 명시하며 테이블을 하나씩 붙여 나가는 구조이지만, Oracle은 가져오고자 하는 컬럼이 포함된 모든 테이블들을 일단 FROM절에 기술하고 WHERE절에 join조건을 명시함.

Outer Join시 조인 당하는 테이블에 (+)표시를 해주어 조인 테이블을 구분해준다. 여기서 Oracle문법으로는 FULL OUTER JOIN이 불가능함.

6. Static SQL과 Dynamic SQL차이

-Static SQL

String 형 변수에 문자를 받지 않고 코드 사이에 직접 문자열을 기술한 SQL문을 말함

다른 말로 ‘Embedded SQL’이라고 함

예시

int main()

{

printf("사번을 입력하십시오 : ");

scanf("%d", &empno);

EXEC SQL WHENAVER NOT FOUND GOTO notfound;

EXEC SQL SELECT ENAME INTO :ename

FROM EMP

WHERE EMPNO = :empno;

printf("사원명 : %s.\n", ename);

notfound:

printf("%d는 존재하지 않는 사번입니다. \n", empno);

}

-Dynamic SQL

String 형 변수에 문자를 받아 동작하는 SQL문을 말함.

String 변수를 사용하므로 조건에 따라 SQL문을 동적으로 바꿀 수 있고, 런타임 시 사용자로부터 SQL문의 일부 또는 전부를 입력 받아서 실행 할 수도 있음.

따라서 PreCompile시 Syntax, Semantics를 확인할 수 없음.

Dynamic SQL을 만나면 PreCompiler는 그 내용을 확인하지 않고 넘어감.

예시

int main()

{

char select\_stmt[50] = "SELECT ENAME FROM EMP WHERE EMPNO = :empno";

// scanf("%c", &select\_stmt); --> SQL문을 동적으로 입력 받을 수도 있음

EXEC SQL PREPARE sql\_stmt FROM :select\_stmt;

EXEC SQL DECLARE emp\_cursor CURSOR FOR sql\_stmt;

EXEC SQL OPEN emp\_cursor USING :empno;

EXEC SQL PETCH emp\_cursor INTO :ename;

EXEC SQL CLOSE emp\_cursor;

printf("사원명 : %s.\n", ename);

}

결론적으로 사용자의 입력을 받아 DB를 수정하거나 읽을 때는 Dynamic SQL문을 작성해야 하지만 그렇지 않을 때는 하드 파싱 부하를 최소화하기 위해 Static SQL문을 사용하는 것이 좋다고 생각함.

**7. MySQL의 쿼리최적화(Optimizer)방법 및 힌트**

A. 테이블 정규화

데이터베이스에서 테이블의 데이터 중복성을 피해야 최적화가 잘 됨.

한 테이블과 다른 테이블을 연결하는 규칙을 잘 설계하고 잘 Join을 해야 중복 데이터를 피하고 저장공간, 속도 등을 개선할 수 있다.

B. 올바른 데이터 유형을 사용

테이블에서 적용되는 데이터와 올바른 데이터 타입을 잘 적용해야 공간 낭비를 막고, 속도를 개선시킬 수 있다.

ex) 주민등록번호 같은 숫자 데이터는 varChar보다는 int 필드에 저장하는 것이 더 빠름

D. Explian 키워드를 사용하여 쿼리 분석

검색 등을 사용할 때 ‘Explain’ 키워드를 사용하여 동작이 어떻게 진행되는지 확인하고 느리게 사용되는 부분이 있는지 확인한다.

E. Union 사용

lile를 이용한 와일드카드 검색을 많은 열에서 사용하면 느려질 수 있다.

그래서 같은 부분을 검색할 때 2열에서 like문 검색을 하지 말고 union문을 활용하여 동일한 쿼리를 최적화할 수 있다.

***-SQL 힌트***

SQL 튜닝의 핵심 부분으로 일종의 지시 구문이며 SQL에 포함되어 쓰여짐. Optimizer 의 실행 계획을 원하는 대로 바꿀 수 있음. DataBase의 Optimizer는 항상 최선의 실행계획을 수립할 수 없어서 테이블이나 인덱스의 잘못된 실행계획을 개발자가 직접 바꿀 수 있게 도와주는 것.

-특징

실행 계획 제어, 에러 발생 안함, 선택 또는 취소 가능, 다양한 종류의 힌트 업데이트

-종류

1. STRAIGHT\_JOIN

여러 테이블 조인시 조인순서를 from의 순서대로 허용해준다.

2. USE INDEX / FORCE INDEX / IGNORE INDEX

인덱스 사용/강제/무시

3. SQL\_CACHE / SQL\_NO\_CACH

조회된 결과를 재사용하기 위해 쿼리 캐시에 선택적으로 저장 할 수 있음.

4. SQL\_CALC\_FOUND\_ROWS

limit절과는 상관없이 조건에 일치하는 모든 레코드를 검색해 결과가 얼마나 되는지 계산함. 실제 결과는 limit에 제한된 건수만 계산함

5. FOUND\_ROWS

직전 쿼리에서 검색된 결과 row 수를 반환함

쿼리 옵티마이저의 작업을 사용하면 검색속도나 데이터 용량 등을 줄일 수 있다. 이 수준의 차이는 적은 테이블 수는 차이가 크게 나지 않지만, 대용량의 DB에서는 큰 차이가 날 것이다. 이는 큰 회사 등에서 사용하는 DB에서는 중요하게 다루어야 할 부분이라고 생각한다.

참고문헌 및 링크

<https://victorydntmd.tistory.com/129>

<https://mommoo.tistory.com/62>

<http://jason-heo.github.io/mysql/2014/03/05/char13-index-types.html>

<https://velog.io/@ansrjsdn/MySQL-INDEX-%EB%9E%80>

<https://ggmouse.tistory.com/131>

<https://www.w3bai.com/ko/sql/sql_wildcards.html>

<https://velog.io/@datata29/SQL-%EC%99%80%EC%9D%BC%EB%93%9C-%EC%B9%B4%EB%93%9C-%EB%AC%B8%EC%9E%90%EB%A5%BC-%EC%9D%B4%EC%9A%A9%ED%95%9C-%ED%95%84%ED%84%B0%EB%A7%81>

<http://www.incodom.kr/DB_-_%EB%8D%B0%EC%9D%B4%ED%84%B0_%ED%83%80%EC%9E%85/ORACLE>

<http://www.incodom.kr/DB_-_%EB%8D%B0%EC%9D%B4%ED%84%B0_%ED%83%80%EC%9E%85/MYSQL>

<https://docs.oracle.com/cd/E12151_01/doc.150/e12155/oracle_mysql_compared.htm#BABGACIF>

<https://energ.tistory.com/entry/Join-%EC%A2%85%EB%A5%98>

<http://wiki.gurubee.net/display/DBSTUDY/Static+vs.+Dynamic+SQL>

<https://thefunky-monkey.com/page-364/mysql-mariadb-2/>

<https://taes-k.github.io/2019/11/22/mysql-hint/>

<https://incheol-jung.gitbook.io/docs/q-and-a/db/sql-hint>