CUBRID 전환 가이드

from ORACLE

2023-12

기술본부

목차

1. 개요

1.1 목적

ORACLE 사용자들이 CUBRID 로의 이전을 보다 쉽게 하기위해 만들어졌으며, ORACLE 과 CUBRID 사용시 차이점을 주로 정리하였으며, 일반적으로 사용되는 함수들 중에서 차이가 있는 부분에 대하여 정리하여, 개발자들의 불편을 최소화할 수 있도록 하였습니다. 또한 함수의 경우, 모든 부분을 다루기는 어려우므로 관련 함수의 지원 여부에 대하여는 매뉴얼(www.cubrid.org/manuals)을 참조하시면 됩니다.

이하 문서의 편의를 위해 존칭은 생략한다.

1.2 사용환경

CUBRID : 11.2 이상

JAVA : 1.8 이상(Stored Procedure 사용시, 데이터베이스 서버 환경, 응용환경 아님)

2. CUBRID 기능

CUBRID 는 통상적인 RDBMS 구조를 가지고 있으며, 지원되는 주요 기능들에 대하여 정리하였다.

구분	기능
SQL	ANSI SQL(SQL-92 기준, SQL-99/2003 호환)
용량	테이블 개수, 레코드 건수 제약 없음
data type	int, bigint, numeric(number), char, varchar, bit varying, etc 외부에 저장되는 LOB(CLOB, BLOB)
charset	문자단위의 문자 처리(length, substring 등) UTF8, EUC-KR 등 지원
transaction	record based locking commit/rollback/savepoint 별도의 트랜잭션 시작 구문없이, 임의의 질의 수행 시 트랜잭션 시작됨 auto commit 은 언어별 설정에 따름
Backup & Recovery	on-line(hot)/off-line(cold) backup Full/Incremental backup Time based recovery(시점 복구) Full recovery (반드시 백업이 존재해야 복구 가능)
Miscellaneous	CTE(Common Table Expressions) Query plan/result Cache Partition

	data encryption(MD5, SHA1, SHA2) TDE(Transparent Data Encryption) 테이블 암호화 Stored Procedure(JAVA) 패킷 암호화 (SSL/TLS(Secure Socket Layer/Transport Layer Security)
HA	자체 기본 지원 별개의 데이터베이스(shared nothing)로 스토리지 장애에 유연함
Sharding	scale out 형태의 데이터 분산(수평분할)
DBlink	CUBRID, Oracle, MySQL 의 데이터베이스 조회 기능
API	JDBC, ODBC, PHP,.NET, C-API, Ruby, Phython, etc
미지원	Temporary Table, Parallel Query, Materialized View

3. 데이터베이스 생성

3.1 데이터베이스 이름

- **대소문자 구분**하므로, 생성시 대소문자에 주의하여야 한다. 응용에서 연결시에도 대소문자를 구분한다.

3.2 데이터베이스 생성

- 1 개의 데이터베이스에 1 개의 인스턴스 만을 가진다. 따라서 여러 개의 인스턴스가 필요한 경우 여러 개의 데이터베이스를 생성해야 한다.
- 생성 후 바로 백업을 하는 것이 좋다. 백업이 있어야만 원하는 시점으로 복구를 할 수가 있기 때문이다.
- 영구 볼륨(테이블 스페이스)은 자동 추가되므로, 별도로 지정할 필요는 없다.
- 현 데이터의 보전 연한 및 향후 추가될 데이터의 양을 산정하여 적절한 여유 공간을 추가하여 데이터베이스의 크기를 산정한다.
- **데이터/인덱스, 질의처리공간(temp) 별도 생성**해주어야 한다. 인덱스 공간도 산정을 통해 계산하는 것이 좋으며, 그렇지 못한 경우 데이터의 20~30%정도로 산정할 수 있다. 질의 처리 공간은 통상적으로 데이터의 10% 수준으로 생성한다.
- 다음의 예는 데이터베이스 mydb 를 /DB 아래, 데이터베이스 로그를 /DB/log 에 생성하고, 데이터가 저장될 공간은 100G 로 생성한다. 생성 명령에 대한 자세한 설명은 매뉴얼을 참고한다.

```
#데이터베이스 생성
$ cubrid createdb -F /DB -L /DB/log mydb ko_KR.utf8
# 데이터가 저장될 공간 100G 생성
$ cubrid addvoldb -S -db-volume-size=100G mydb
# 질의 처리공간 생성
$ cubrid addvoldb -S -p temp -db-volume-size=10G mydb
```

3.3 데이터베이스 사용자계정/암호 생성 및 서비스 구동

- 데이터베이스를 생성 후 관리자 계정인 dba(시스템 관리자)계정으로 접속하여 응용에서 사용할 사용자 계정을 생성하고 패스워드를 부여한다. CMT 는 사용자 계정 이관지 않으므로 사용자 계정은 직접 생성하여야 한다.응용에 필요한 테이블 생성 등의 작업을 할 수 있다.
    - ◆ 데이터베이스 생성시 dba 계정은 암호가 없으므로, 암호를 부여하는 것을 권장한다.
- 사용자가 제대로 생성되었는지 확인하기 위해 csql 을 이용하여 접속을 해본다.시 사용자 명은 대소문자를 구분하지 않지만, 데이터베이스 이름과 암호는 대소문자를 구분한다.
- 보안상 dba 계정에 대한 암호 설정을 권고한다.

예) 리눅스 터미널(curbrid 계정)에서 csql 인터프리터로 DB 에 접속하여 관리자 계정인 dba 의 패스워드 생성 후 사용자 계정 및 패스워드 생성

```
$ csql -u dba -S mydb
csql> alter user dba password ‘dba321’;
csql> create user myuser password ‘myuser321’;
csql>;exit
```

4. 데이터베이스 이관

데이터베이스 이관은 CMT(CUBRID Migration Toolkit)을 이용하여 이관할 수 있으며, https://www.cubrid.org/downloads 에 CUBRID Tools 의 CUBRIDMigration Toolkit 에서 다운 받을 수 있다.

CMT 는 1:1 이관(이전 전후의 스키마(테이블, 인덱스)와 데이터가 동일)과 전체 이관만이 가능하다.

- 이관 전후의 스키마/데이터가 동일하며 증분 이관은 원본에 trigger 를 이용하여야만 가능하다. 그러나 서비스 중인 원본 데이터베이스에 trigger 를 추가하는 것은 사실상 불가능하므로, 데이터베이스 이관은 증분 이관이 어렵다.
- 이관 후 업무를 위해 데이터베이스(스키마/데이터)에 변경이 가해졌다면, 재 이관 시 변경사항이 사라지게 되므로 재 이관 후 변경 작업을 다시 할 수 있도록 준비(스키마 변경 SQL, 데이터 변경 SQL 또는 프로그램)가 되어 있어야 한다.
- 업무에 맞게 변경하는 것은, 업무를 파악하고 업무에 맞게 조정되어야 하므로 이관 후 별도의 과정을 통해 조정하여야 한다.

CMT 를 이용하여 전환 가능한 대상 데이터베이스는 ORACLE, MySQL, MariaDB, MS-SQL 5.0 이상이 가능하다.

본 문서는 다음의 내용을 기준으로 작성되었다.

1. 이관 원본: oracle 10g,
   A. SID: xe, DB 계정: db_user1, db_user2, db_user3
   B. 설치 서버: localhost
2. 이관 대상: CUBRID 11.3
   A. 데이터베이스 이름: mydb, DB 계정: myuser
3. 이관도구: CMT 11.2
4. 이관 방안: 아래 스키마 및 관련 데이터를 CUBRID DB 의 1 개 계정으로 이관.
   A. oracle 사용자 db_user1, db_user2 의 스키마 → CUBRID myser 계정
   B. oracle 사용자 db_user2 가 db_user3 으로부터 grant 받은 일부 테이블 → CUBRID myuser 계정
5. 사용된 스키마 정보는 다음과 같다.

스키마명	스키마 정보

db_user1	CREATE TABLE "user" (
	  id int PRIMARY KEY,
	  name char(10) NOT NULL,
	  addr varchar(100),
	  birth DATE DEFAULT sysdate,
	  photo blob
	);
	CREATE INDEX idx1 ON "user"(name);
	
	CREATE TABLE dept (
	  id int PRIMARY KEY,
	  name varchar(100),
	  user_id int NOT NULL,
	  "COMMENT" clob,
	  FOREIGN KEY (user_id) REFERENCES "user"(id)

	);
	
	CREATE SEQUENCE user_seq;
	
	CREATE VIEW v_get_dept_user AS
	  SELECT d.id id, d.name name, u.name user_name
	  FROM dept d, "user" u
	  WHERE d.user_id = u.id;
	
	CREATE OR REPLACE PROCEDURE get_dept_info (p_id NUMBER)
	IS
	    v_dname VARCHAR2;
	    v_uname VARCHAR2;
	BEGIN
	    SELECT d.name, u.name
	    INTO v_dname, v_uname
	    FROM dept d, "user" u
	    WHERE d.user_id = u.id AND d.id = p_id;
	
	    DBMS_OUTPUT.PUT_LINE('DEPT info retrieved successfully:');
	    DBMS_OUTPUT.PUT_LINE('ID: ' || p_id);
	    DBMS_OUTPUT.PUT_LINE('DEPT Name: ' || v_dname);
	    DBMS_OUTPUT.PUT_LINE('USER Name: ' || v_uname);
	EXCEPTION
	    WHEN NO_DATA_FOUND THEN
	        DBMS_OUTPUT.PUT_LINE('Record not found for ID: ' ||
	   p_id);
	    WHEN OTHERS THEN
	        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
	END get_dept_info;

db_user2	CREATE TABLE "user" (
	   id int PRIMARY KEY,
	   name char(10),
	   addr varchar(100),
	   birth DATE DEFAULT sysdate,
	   photo blob
	);
	COMMENT ON TABLE "user" IS '그룹원정보';
	COMMENT ON column "user".name IS '그룹원 이름';
	
	CREATE INDEX idx1 ON "user"(name);
	
	CREATE TABLE "group" (
	   id int PRIMARY KEY,
	   name varchar(100),
	   user_id int NOT NULL,
	   "COMMENT" clob,
	   FOREIGN KEY (user_id) REFERENCES "user"(id)
	);
	
	create synonym school for db_user3.school;

	`CREATE VIEW v1 AS SELECT name FROM school;`

db_user3	`CREATE TABLE teacher (`
	`  id int PRIMARY KEY,`
	`  name char(10),`
	`  addr varchar(100),`
	`  birth DATE DEFAULT sysdate,`
	`  workday number DEFAULT CAST(to_char(sysdate, 'ddd') as number),`
	`  photo blob`
	`);`
	`CREATE INDEX idx1 ON teacher(name);`
	
	`CREATE OR REPLACE TRIGGER tr1  BEFORE INSERT ON teacher`
	`BEGIN`
	`  IF birth = sysdate`
	`  THEN workday = 0`
	`  ELSE workday = workday - 1`
	`  END IF`
	`END;`
	
	`CREATE TABLE school (`
	`  id int PRIMARY KEY,`
	`  name varchar(100),`
	`  tid int NOT NULL,`
	`  "COMMENT" clob,`
	`  FOREIGN KEY (tid) REFERENCES teacher(id)`
	`);`
	
	`grant SELECT ON school to db_user2;`
	Trigger 는 예를 위한 것으로 trigger 생성후 insert 시 오류 발생.
	테스트를 위해 insert 후 trigger 만 생성

4.1 스키마/데이터 export 방법

사용법은 비교적 간단하며, java 와 oracle jdbc driver(ojdbc6 권장) 가 준비되어 있어야 한다. 내려받은 CMT 는 적절한 위치에 압축을 풀어주면 된다. 기본적으로 C:₩cubridmigration 에 설치된 것으로 가정하여 진행한다.

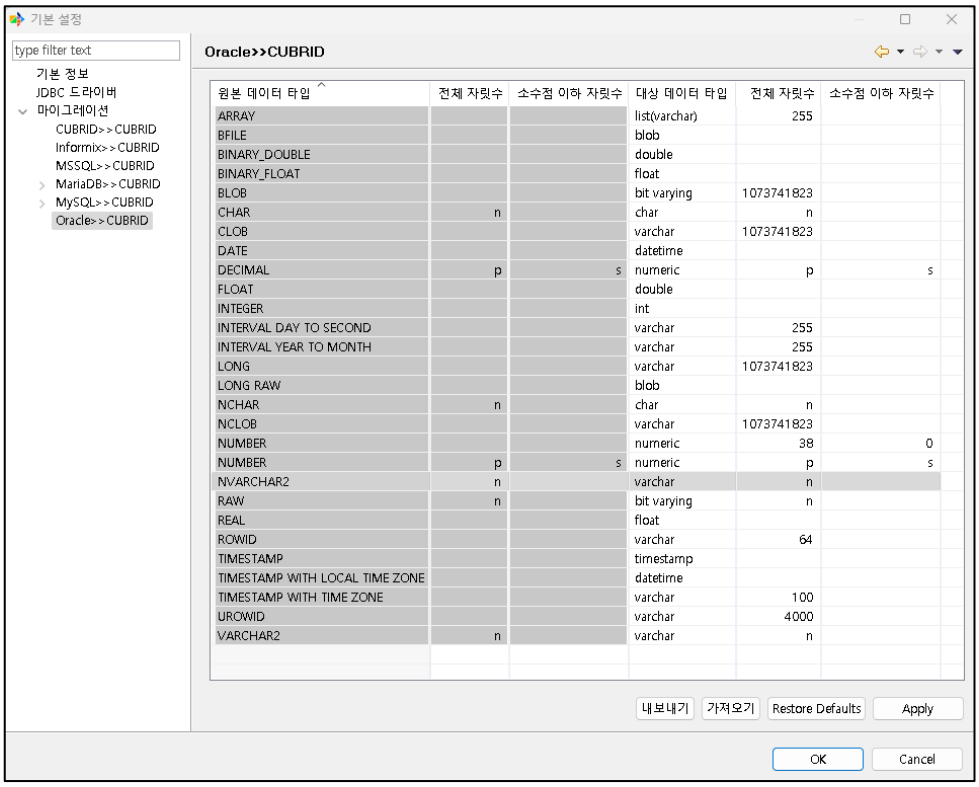
※ CMT 에서 이관시 한 계정의 스키마에 대하여 이관이 가능하며, 여러 계정의 스키마를 동시에 이관할 수는 없다. 따라서 다음과 같이 내용에 맞게 단계적으로 이관한다.

1. db_user1 과 db_user2 의 스키마를 myuser 라는 하나의 계정으로 이관하여야 하므로, db_user1 을 먼저 이관하고, db_user2 를 다시 이관한다.
2. db_user2 가 db_user3 로부터 select 권한을 가진 일부 스키마는 db_user2 이관시 같이 이관할 수 있다.
3. 따라서 db_user1 의 스키마를 1 회, db_user2 의 스키마와 db_user2 가 grant 받은 db_user3 의 일부 스키마에 대하여 1 회, 총 2 회에 걸쳐 이관을 하여야 한다.

1 기본 설정 수정

HA 환경일 경우: LOB 타입을 데이터베이스 내부에 저장하기 위해 대상 데이터 타입을 변경하여야 한다. CMT 상단 우측의 “기본 설정”을 클릭하고, 좌측의 마이그레이션 아래 Oracle>>CUBRID 를 클릭한다.

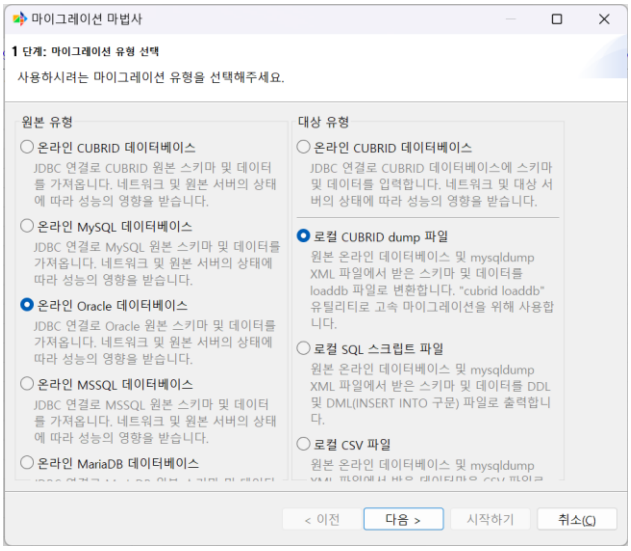
- CLOB, LONG : varchar 1G
- BLOB : Bit varying 1G (bit 이므로 실 데이터 기준 128M 미만)



원본 데이터 타입	전체 자릿수	소수점 이하 자릿수	대상 데이터 타입	전체 자릿수	소수점 이하 자릿수
ARRAY			list(varchar)	255	
BFILE			blob		
BINARY_DOUBLE			double		
BINARY_FLOAT			float		
BLOB			bit varying	1073741823	
CHAR	n		char	n	
CLOB			varchar	1073741823	
DATE			datetime		
DECIMAL	p	s	numeric	p	s
FLOAT			double		
INTEGER			int		
INTERVAL DAY TO SECOND			varchar	255	
INTERVAL YEAR TO MONTH			varchar	255	
LONG			varchar	1073741823	
LONG RAW			blob		
NCHAR	n		char	n	
NCLOB			varchar	1073741823	
NUMBER			numeric	38	0
NUMBER	p	s	numeric	p	s
NVARCHAR2	n		varchar	n	
RAW	n		bit varying	n	
REAL			float		
ROWID			varchar	64	
TIMESTAMP			timestamp		
TIMESTAMP WITH LOCAL TIME ZONE			datetime		
TIMESTAMP WITH TIME ZONE			varchar	100	
UROWID			varchar	4000	
VARCHAR2	n		varchar	n	

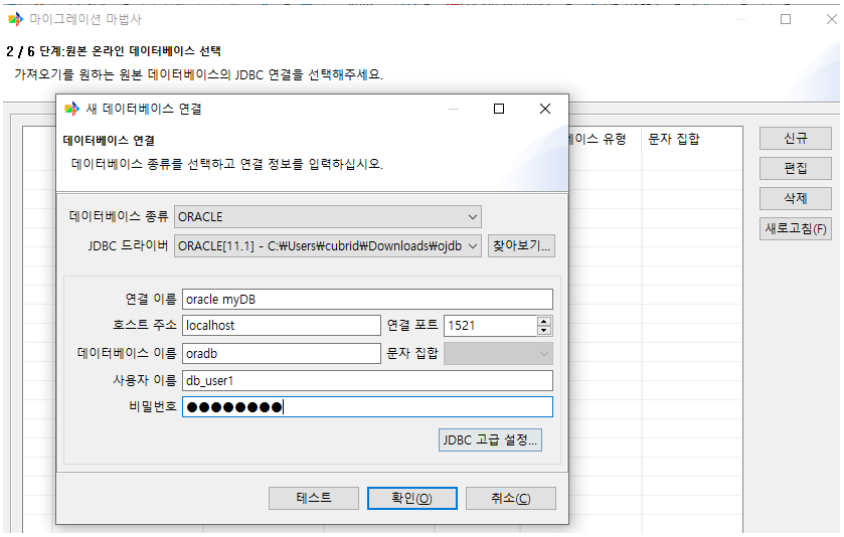
2 마이그레이션 유형 선택

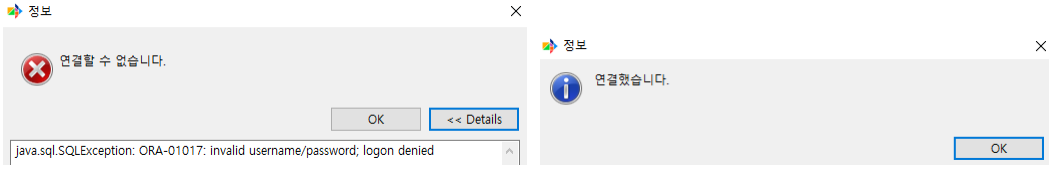
좌측 상단의 “새 마이그레이션”을 클릭하고, 온라인 Oracle 데이터베이스에서 로컬 CUBRID dump 파일로 저장한다. Online 으로 CUBRID 에 직접 입력 가능하나 작업중 에러 발생시 에러 부분만 복구하여 다시 진행하기가 어려워 처음부터 다시 진행하여야 하므로, dump 파일로 저장 후 dump 파일을 CUBRID 에 로딩 하는 형태로 진행하는 것이 안전하다.



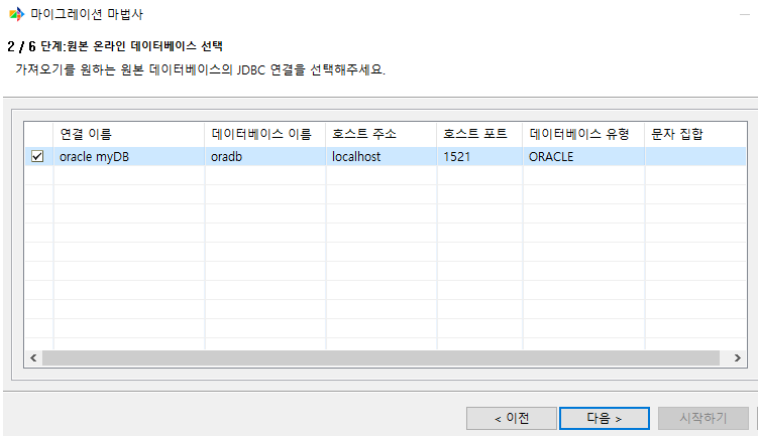
3 소스(원본) 데이터베이스 연결 설정

- 오라클 연결 정보 등록: 신규 클릭 후 등록
  - 연결이름은 사용자가 확인가능한 이름을 자유로이 지정하면 되며, 호스트 주소는 호스트 네임으로도 사용 가능하다.
  - 데이터베이스 이름은 SID 를 입력하고, 사용자 이름은 db_user1 을 먼저 이관할 것이므로 db_user1 을 입력한다.
  - 테스트를 클릭하여 연결이 정상적으로 되는지 반드시 확인. 연결 실패 시 상세정보를 확인하여 연결 문제 해결하여야 한다.





- 정상적으로 연결이 되면, 생성한 연결을 선택하고, 다음을 클릭한다.

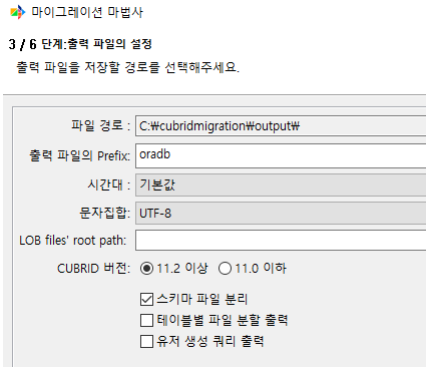


4 출력 파일의 설정

출력 파일의 위치와 접두사를 지정할 수 있으며, 스키마 파일 분리, 테이블별 파일 분할 출력, 유저 생성 쿼리 출력 옵션을 통해 지정 가능하다.

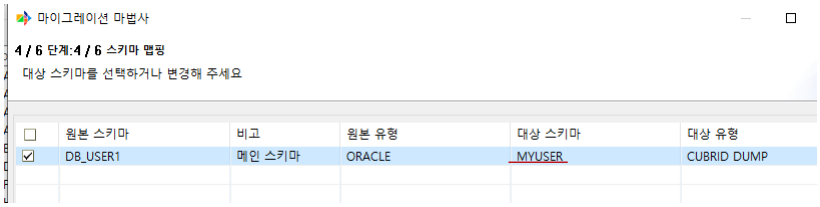
- 스키마 파일은 분리하는 것이 작업 효율을 위해서 좋다. 데이터가 많은 경우 키나 인덱스를 데이터 입력 후에 생성하는 것이 성능상 유리하며, 사용자간 grant 등이 있을 경우 분리하여 사용자별 스키마를 등록 후 grant 등을 진행하여야 하므로 분리하는 것이 좋다.
  - 분할된 파일에 대한 설명은 뒤의 import 부분에서 설명한다.
- 테이블별 파일 분할은 데이터 파일을 테이블 별로 생성하도록 하는 것이다. 이기종 데이터베이스로부터 import 시 에러가 발생하는 경우 에러 발생한 테이블부터 재작업을 하는 것이 작업 시간을 줄일 수 있으므로 분리하는 것이 좋다.
- 유저의 경우 TO-BE 에서 변경되는 경우가 많아 일반적으로 유저 생성을 미리하고 진행하는 것이 좋지만, 필요시 생성을 위한 SQL 문이 포함된 화일을 생성할 수 있다.

DB 버전은 사용할 CUBRID 의 버전이며 사용버전에 맞게 선택한다. 11.2 미만은 사용자 스키마를 지원하지 않는다.



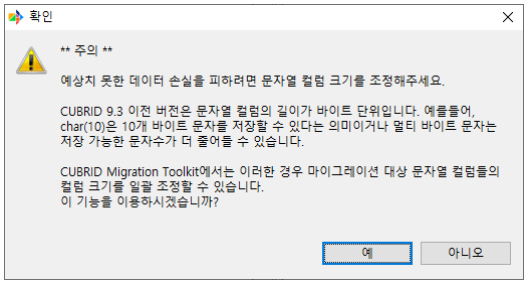
5 스키마 맵핑

원본 스키마와 대상 스키마명을 확인하고, 대상 스키마 명이 myuser 로 변경되므로 대상 스키마명을 myser로 수정하고 다음을 클릭한다. db_user1 으로 로그인하였으므로 db_user1 에서 접근 가능한 스키마만 보인다.

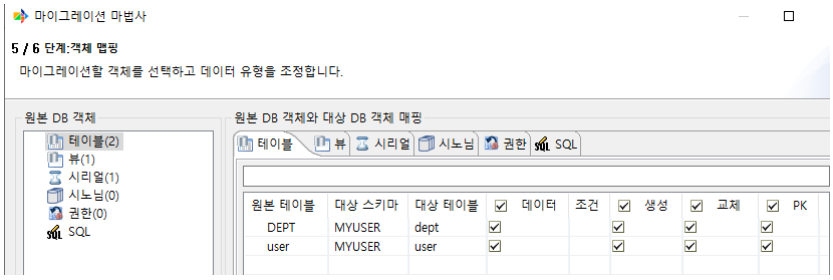


6 객체 매핑

"예상치 못한 데이터 손실을 피하려면 문자열 컬럼 크기를 조정해주세요" 주의사항에 "아니오"를 클릭한다. 9.3 이상 버전에서는 문자열 단위로 문자길이를 계산하므로 이 기능은 필요 없다.

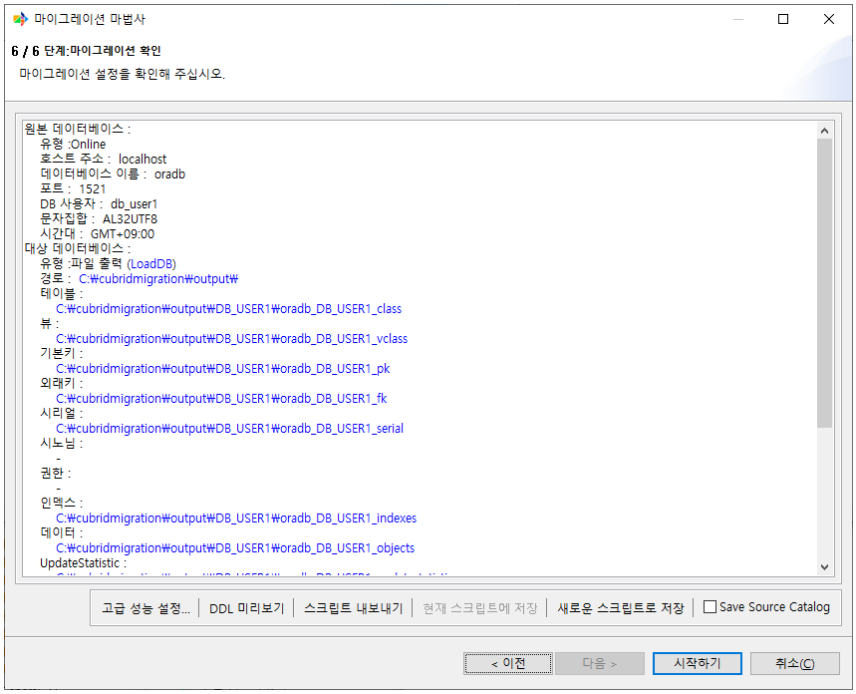


CUBRID 에서 스키마 정보는 모두 소문자로 적용되며, 사용자 정보는 모두 대문자로 적용된다. 질의 사용시 대소문자는 구분하지 않는다. 실제 이관할 테이블 뷰 등을 선택한다. 일부 테이블이나 뷰를 이관할 경우 오른쪽 화면에서 테이블 탭이나 뷰 탭을 선택하고 원하는 테이블/뷰 만 선택하면 된다. 스키마만 이관 시 데이터에 체크를 해제하면 된다.



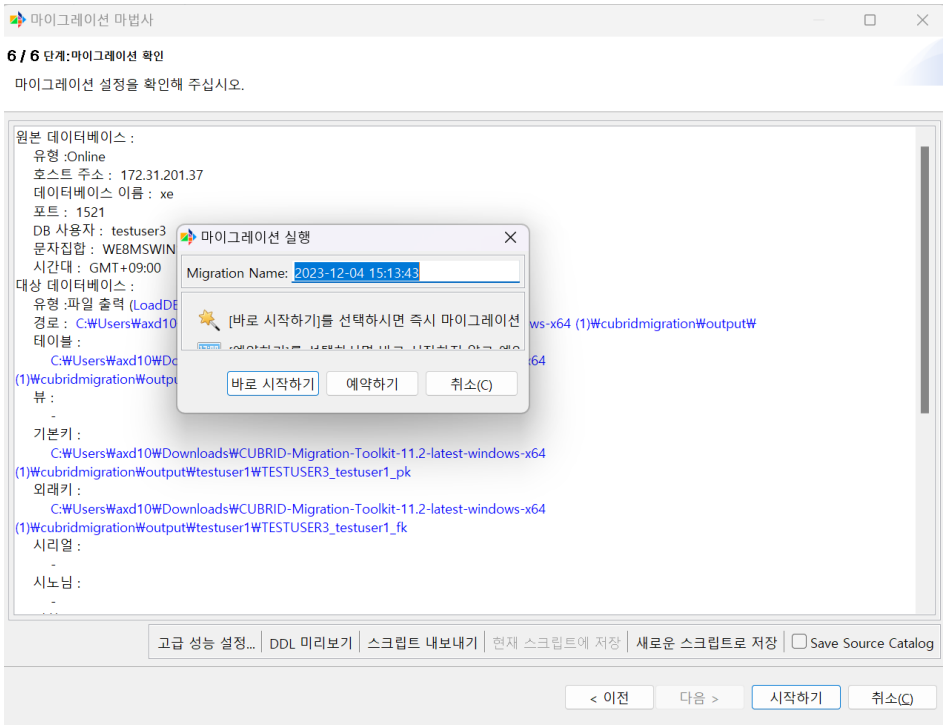
7 마이그레이션 설정 확인

"다음"을 클릭하고 계속 진행여부에서 "Y"를 클릭하면 현재 설정된 마이그레이션 정보가 보여 진다. 대상 데이터베이스의 출력 경로를 보면 앞서 지정한 폴더에 대상 스키마 명으로 폴더를 만들고 그 아래 파일들이 만들어지는 것을 확인할 수 있다.

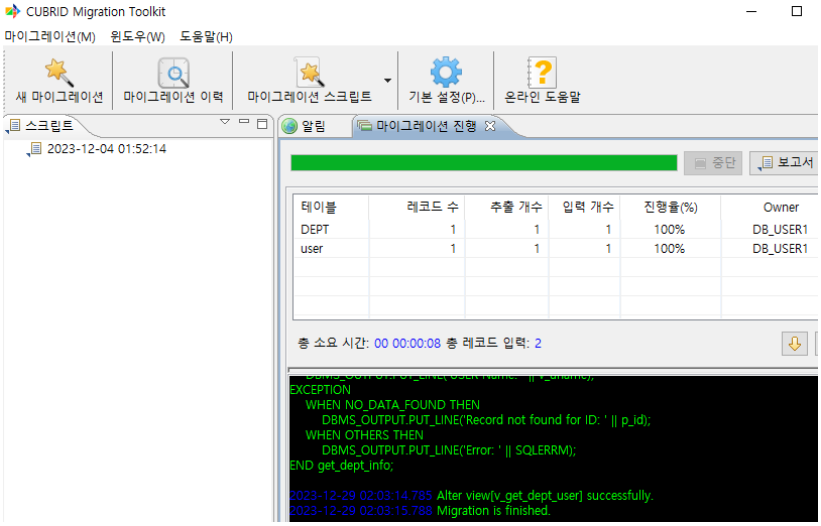


8 마이그레이션 진행

"시작하기"를 클릭하면 마이그레이션 이름 입력창이 나오는데, 이곳에 적당한 이름을 넣어주거나 기본값 그대로 사용하여도 된다. 추후 동일한 조건으로 재 이관 시 사용할 수 있다.



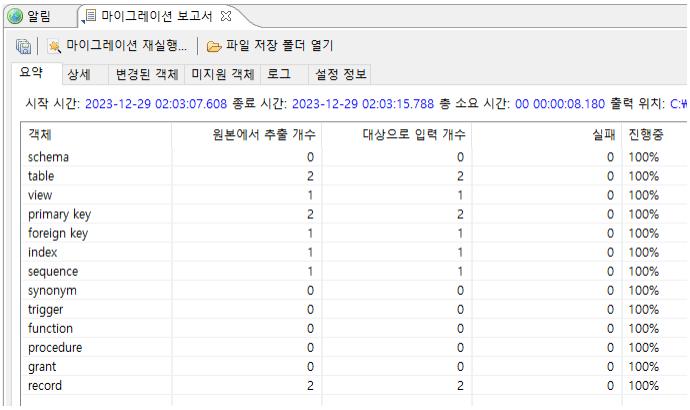
"바로 시작하기"를 클릭하면 마이그레이션이 진행된다. 각 테이블에 대하여 총 개수와 가져온 개수와 진행율을 보여준다. 추출개수가 원본 데이터베이스로부터 가져온 개수이며, 입력 개수가 대상 데이터베이스로 입력한 개수이다. 본 예에서는 파일로 만들었으므로 파일에 쓰여진 개수로 보면 된다.



9 완료

완료 후 보고서를 볼 수 있으며, 만약 오류가 발생한 경우 붉은 색으로 표시되며 진행률이 100%가 아닌 부분 중 어느 한 곳(동시에 여러 테이블을 이관하므로)에서 에러가 발생한 것이며, 상세를 클릭해서 이전통계 탭을 보면 테이블별 진행율을 확인할 수 있다.

이관 전후 데이터의 비교는 테이블 개수나 레코드 개수만으로 가능하며, 테이블이나 레코드 개수는 요약 화면을 참고하면 된다.



객체	원본에서 추출 개수	대상으로 입력 개수	실패	진행중
schema	0	0	0	100%
table	2	2	0	100%
view	1	1	0	100%
primary key	2	2	0	100%
foreign key	1	1	0	100%
index	1	1	0	100%
sequence	1	1	0	100%
synonym	0	0	0	100%
trigger	0	0	0	100%
function	0	0	0	100%
procedure	0	0	0	100%
grant	0	0	0	100%
record	2	2	0	100%

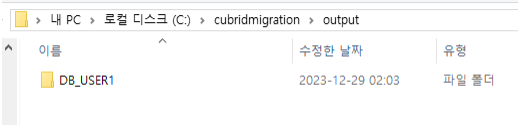
10 Stored procedure/function (PL/SQL) 및 트리거

Stored procedure/function 은 현재 java 를 이용하도록 하고 있어, PL/SQL 은 java 로 변환하여 등록하여야 한다. Java 작성 방법은 매뉴얼을 참고한다. CUBRID 11.4 부터는 PL/CSQL 을 지원할 예정이다.

PL/SQL 내용은 위 보고서의 미지원 객체에 있으므로, 해당 내용을 그대로 가져가거나 ORACLE 에서 PL/SQL 부분을 꺼내면 된다.
또한 트리거에서 PL/SQL 이 사용되며, 문법도 상이한 부분이 있어 트리거는 ORACLE 에서 추출하여 CUBRID 에 맞게 보완하여 등록하여야 한다. PL/SQL 을 사용하는 부분은 필요하면 java SP 로 작성하여 등록할 수 있다.

11 Dump 파일 서버로 전송

dump 파일을 대상 CUBRID 서버에 ftp 등을 이용하여 올려 준다. 출력으로 지정한 디렉토리에 대상 스키마 명의 폴더에 있는 파일을 폴더채로 올려주면 된다.

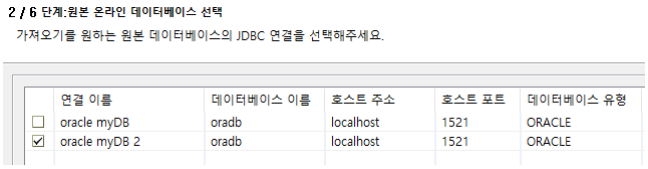


12 db_user2 추가 이관

앞서 이관 절차는 모두 설명하였으므로 db_user1 이관시와 다른 점만 설명한다.

12.1 마법사 2/6 원본 선택

기존 설정을 편집하거나 신규로 새로이 추가한다.



12.2 마법사 3/6 출력 선택

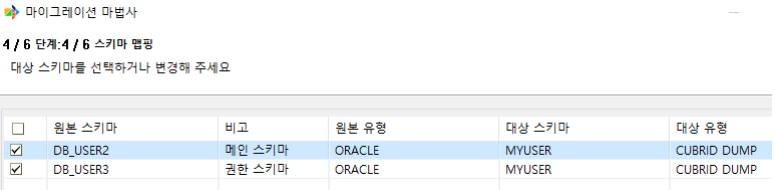
12.2.1 출력 위치는 동일하게 하여도 된다. 원본 스키마 명이 달라 별도의 폴더에 스키마/데이터 파일이 생긴다. 물론 별도의 위치로 지정하여도 된다.

예1) DB_USER1/oradb_DB_USER1_class (oradb 의 db_user1 스키마의 테이블 DDL 파일)

예2) DB_USER2/oradb_DB_USER2_class (oradb 의 db_user2 스키마의 테이블 DDL 파일)

12.3 마법사 4/6 스키마 매핑

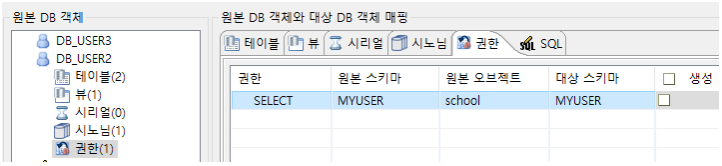
db_user1 과는 다르게 다른 스키마도 보인다. 이는 db_user3 에서 일부 테이블에 대한 조회 권한을 받았기 때문에 해당 테이블에 대한 정보가 보인다. 이 테이블도 이관하기로 하였으므로 2 개의 스키마를 모두 선택하고, 대상 스키마는 myuser 로 수정한다.



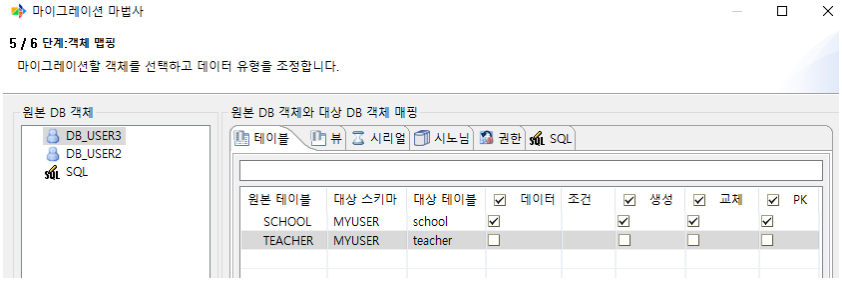
12.4 객체 매핑

12.4.1 권한 확인: db_user3 의 모든 테이블이 아닌 db_user2 가 select 권한을 가진 테이블만 이관할 것이므로, 어느 테이블인지 확인한다. 생성은 grant 문이 들어가 있는

SQL 파일을 만들어 주는 것인데, CUBRID 의 한 계정으로 모두 들어가므로 grant 할 필요가 없다. 만약 별도의 계정으로 간다면 생성 하여야 한다. 내용에 대한 설명은 뒤의 유의사항에 있다.



12.4.2 grant 받은 테이블 선택: db_user3 의 스키마에서 school 테이블만 선택한다.



12.5 dump 파일 서버로 전송

5. CUBRID import

CMT 를 이용하여 dump 받은 파일을 서버로 전송한 뒤 해당 파일들을 이용하여 Import 는 다음과 같이 진행한다. Export 받은 모든 파일은 output 디렉토리에 있는 것으로 가정한다.

import 진행 중 오류 발생 시 뒤의 “이관 시 유의사항”을 참고한다.

- 스키마 등록은 csql(CUBRID SQL Interpreter) 을 이용한다.
  - 등록 중 오류가 발생할 수 있으므로 오류를 수정하면서 진행하여야 하므로 csql 을 이용한다. 접속 시 --no-auto-commit 옵션을 사용하여 접속한다.
  - 오류 발생시 rollback 하거나, rollback 없이 정상 처리된 부분까지는 삭제하고, 오류가 발생한 부분에 대하여 수정하거나 또는 수정이 불가능한 경우 주석 처리하고 계속 진행한다.
  - 모든 내용이 정상적으로 등록되면 commit 하고 종료한다.
- 데이터 로딩은 loaddb 로 진행한다.

5.1 스키마/데이터 import 방법

1 USER 생성

데이터베이스 생성 및 사용자 생성은 위의 데이터베이스 생성 및 사용자 생성을 참고한다. 서비스 구동은 일단 하지 않는다.

2 테이블 생성

2.1 테이블 생성을 위한 DDL 은 _class 파일에 있으므로 이 파일을 읽어서 실행한다. db_user1 의 테이블 정보 파일을 먼저 읽어서 실행하고, 정상적으로 수행되면 commit 을 수행하고, 종료한다.

```
$ cd output/DB_USER1
$ csql -u myuser -p 'myuser321' -S --no-auto-commit mydb
CUBRID SQL Interpreter

Type `;help' for help messages.
csql> ;read oradb_DB_USER1_class

The file has been read into the command buffer.
csql> ;run
Execute OK. (0.004030 sec)
Execute OK. (0.002626 sec)

2 command(s) successfully processed.
```

```
csql> ;commit

Committed.
csql> ;exit
```

3 데이터 로딩

loaddb 라는 유틸리티를 이용하여 데이터를 로딩한다. 성공적으로 데이터베이스에 로딩된 총 레코드의 개수는 아래 2 건임을 알 수 있다. 이를 이용하여 이관 전후의 데이터 양을 비교할 수 있다.

```
$ cubrid loaddb -S -u myuser -p 'myuser321' -v -c 10000 --no-oid --no-statistics -d oradb_DB_USER1_objects mydb

Start object loading.
Class MYUSER.dept
MYUSER.dept 1 instances
Class MYUSER.user
MYUSER.user 1 instances
Total 2 object(s) inserted, 0 object(s) failed.

*** Committing the transaction ***

*** Closing the database ***
```

4 기본키/외래키 및 인덱스 등록

csql 을 이용하여 스키마 등록시와 같은 형태로 기본키와 외래키 그리고 인덱스를 등록한다. 내용의 이해를 위해 화면에 표시되는 정보 중 반드시 필요한 것이 아니면 삭제하였다.

```
$ csql -u myuser -p 'myuser321' -S --no-auto-commit mydb
CUBRID SQL Interpreter

Type `;help' for help messages.
csql> ;read oradb_DB_USER1_pk
csql> ;run
csql> ;commit
csql> ;clear
csql> ;read oradb_DB_USER1_fk
csql> ;run
csql> ;commit
csql> ;clear
csql> ;read oradb_DB_USER1_index
csql> ;run
csql> ;commit
```

```
csql> ;clear
```

5 serial 및 view 등록

csql 을 이용하여 serial 및 view 를 등록한다. view 는 view 선언과 관련 질의 부분으로 파일이 나누어져 있다.

```
$ csql -u myuser -p 'myuser321' -S --no-auto-commit mydb
 CUBRID SQL Interpreter

Type `;help' for help messages.
csql> ;read oradb_DB_USER1_serial
csql> ;run
csql> ;commit
csql> ;clear
csql> ;read oradb_DB_USER1_vclass
csql> ;run
csql> ;commit
csql> ;clear
csql> ;read oradb_DB_USER1_vclass_query_spec
csql> ;run
csql> ;commit
csql> ;clear
```

6 db_user2 테이블 등록

6.1 위와 동일한 방법으로 db_user2 의 테이블을 등록한다. 이번엔 에러가 발생하는데 에러의 내용이 user 라는 테이블이 이미 존재한다는 에러이다. 2 개의 스키마가 1 개의 스키마로 합쳐지다 보니 발생할 수 있는 에러이다. 여러 개의 스키마를 하나의 스키마로 이관하는 경우 테이블 명의 중복이 없어야 쉽게 이관할 수 있으며, 이와 같이 중복되는 테이블이 있는 경우 이미 등록된 테이블의 이름을 변경하고 새로운 테이블 및 데이터 로딩이 가능하다. 데이터를 export 한 파일에 스키마 명이 포함되어 있어 이 부분을 수정하기는 매우 어렵기 때문이다.

```
$ cd ../DB_USER2
csql> ;read oradb_DB_USER2_class
The file has been read into the command buffer.
csql> ;ru
Execute OK. (0.003745 sec)
In line 9, column 1,
ERROR: before ' (
[id] int,
[name] char(10),
[addr] varchar(100),
[birth] dat...'
```

```
Class myuser.user already exists.

1 command(s) successfully processed.
csql> ;rollback
csql> ;clear

# db_user1 의 user 테이블 이름과 중복되므로, db_user1_user 로 이름을 변경한다.
csql> rename table [user] as db_user1_user;
Execute OK. (0.001768 sec)
# 다시 테이블 정보화일을 읽어들이고 실행한다.
csql> ;read oradb_DB_USER2_class
csql> ;run
Execute OK. (0.002629 sec)
Execute OK. (0.002538 sec)
2 command(s) successfully processed.
# 정상 수행되었으므로 commit 하고 종료한다.
csql> ;commit
csql> ;exit
```

7 db_user1 과 같은 방법으로 데이터를 로딩하고, 기본키/외래키 와 인덱스 뷰를 등록한다.

7.1 뷰에 질의 등록시 에러가 발생한다. 내용을 보니 school 테이블이 없어 발생하는 것이고, 이 테이블은 db_user3 의 테이블을 grant 받아 사용하고 있었으므로, db_user3 의 테이블을 등록하지 않아 발생하는 것이다. 따라서 db_user3 를 먼저 등록하고 마지막으로 현재 뷰를 등록해야 한다. rollback 하고 종료한다.

```
csql> ;read oradb_DB_USER2_vclass
The file has been read into the command buffer.
csql> ;ru
Execute OK. (0.001504 sec)
1 command(s) successfully processed.
csql> ;cl
csql> ;read oradb_DB_USER2_vclass_query_spec
The file has been read into the command buffer.
csql> ;run
In line 1, column 47,
ERROR: before ' ; '
Unknown class "myuser.school".
csql> ;rollback
csql> ;exit
```

8 db_user3 의 스키마와 데이터를 db_user1 과 같은 방법으로 데이터를 등록한다.

9 db_user2 의 뷰를 다시 등록한다.

```
csql> ;read oradb_DB_USER2_vclass
The file has been read into the command buffer.
csql> ;ru
Execute OK. (0.001504 sec)
1 command(s) successfully processed.
csql> ;cl
csql> ;read oradb_DB_USER2_vclass_query_spec
The file has been read into the command buffer.
csql> ;run
Execute OK. (0.001504 sec)
1 command(s) successfully processed.
csql> ;commit
csql> ;exit
```

6. 이관 시 유의사항

6.1 대량의 데이터 이관 시 mermoy 부족

대량의 데이터이관 시 java memory 부족이 발생할 수 있다. 이를 예방하기 위해 CMT 설치 디렉토리의 cubridmigration.ini 에서 메모리 설정을 다음과 같이 수정한다.

```
Xms1000m
-Xmx4000m
```

6.2 컬럼 Default 에 지원되지 않는 함수 사용 시 오류

컬럼 default 에 지원되지 않는 함수를 사용 시 작은따옴표가 들어가면서 오류가 발생한다.

다음 예시는 스키마 파일 load 할 때 발생한 오류를 보여준다.

컬럼 default 에서 지원되는 함수 목록은 매뉴얼을 참고한다.

1 스키마 파일 실행 시 오류 발생

스키마 파일을 돌렸을 때 오류가 발생시 해당 오류 구문 주석 처리 또는 수정 후 진행한다.

해당 오류의 17 라인으로 이동하여 확인한다.

```
csql> ;ru

In the command from line 17,

ERROR: In line 17, column 0 before ' description available - Waiting',
        'No description ava...'
Syntax error: unexpected 'No', expecting REFERENCES
```

```
 11 ;
 12 CREATE TABLE [testuser1].[nvl_default_table](
 13 [id] numeric(38,15),
 14 [status] varchar(20),
 15 [approval_status] varchar(20) DEFAULT 'Approved',
 16 [description] varchar(1073741823) DEFAULT 'NVL(
 17          'No description available - Waiting',
 18          'No description available'
 19      )'
 20 )
 21 ;
```

2 구문 주석 처리

주석 처리 후 수정 진행한다.

```
12 /*CREATE TABLE [testuser1].[nvl_default_table](
13 [id] numeric(38,15),
14 [status] varchar(20),
15 [approval_status] varchar(20) DEFAULT 'Approved',
16 [description] varchar(1073741823) DEFAULT 'NVL(
17         'No description available - Waiting',
18         'No description available'
19     )'
20 )
21 ;*/
```

3 구문 수정

해당 문제 파악이 될 경우 수정한다.

DEFAULT 'NVL(~)' 구문에서 nvl 함수가 문자로 인식하면서 문제 발생된다.

```
12 CREATE TABLE [testuser1].[nvl_default_table](
13 [id] numeric(38,15),
14 [status] varchar(20),
15 [approval_status] varchar(20) DEFAULT 'Approved',
16 [description] varchar(1073741823) DEFAULT NVL(
17         'No description available - Waiting',
18         'No description available'
19     )
20 )
21 ;
```

4 수정 후 실행

```
csql> ;ru
Execute OK. (0.026012 sec) Committed. (0.012521 sec)
Execute OK. (0.009251 sec) Committed. (0.007387 sec)
Execute OK. (0.009342 sec) Committed. (0.011310 sec)
Execute OK. (0.013547 sec) Committed. (0.007411 sec)
Execute OK. (0.009357 sec) Committed. (0.007296 sec)
```

5 추후에 사용하기 위해 수정한 파일 별도로 저장 필요

```
csql> ;write TESTUSER1_testuser1_class_output

Command buffer has been saved.
```

6.3 예약어 인식으로 인한 오류

예약어가 들어갈 경우 오류가 발생한다. 예약어는 문자로 인식하기 위해서 사용시 “” , ``(백틱, 작은따옴표가 아닌 ~키에 있는) 또는 [] 로 감싸서 사용한다.

일반적으로 타입명, 함수명 등이 예약어가 된다. 자세한 예약어 리스트는 매뉴얼을 참고한다.

다음 예제는 view class load 시 “YEAR” 예약어 인하여 오류 떨어지는 과정을 보여준다.

1 View class 파일 실행 시 오류 발생

오류가 발생시 해당 오류 구문 주석 처리 또는 수정 후 진행한다.
해당 오류의 1 라인으로 이동하여 확인한다.

```
csql> ;read TESTUSER1_testuser1_vclass_query_spec

The file has been read into the command buffer.
csql> ;ru

In line 1, column 35,

ERROR: invalid use of year
 YEAR( expression )


0 command(s) successfully processed.
```

2 구문 수정

"YEAR" 예약어 이므로 문자로 인식이 안되고 있어 [year]로 수정한다.
다음 치환 방법으로 진행하면 변경이 용이하다.

##interactive 모드로 치환 방법 :%s/year/[YEAR]/c

```
ALTER VIEW [testuser1].[example_cube_view] ADD QUERY SELECT [year], status, COUNT(*) AS status_count
FROM example_table_with_year
GROUP BY CUBE([year], status);
ALTER VIEW [testuser1].[example_status_alpha_view] ADD QUERY SELECT
    id,
    [year],
    status,
    REGEXP_SUBSTR(status, '[[:alpha:]]+') AS extracted_alpha
FROM example_table_with_year;
ALTER VIEW [testuser1].[instr_v] ADD QUERY select instr(id,'a',1,1) AS id from test_table1;
ALTER VIEW [testuser1].[test_table1_calculated_view] ADD QUERY SELECT
    id,
    name,
    get_value_by_name(name) AS calculated_value
FROM test_table1;
```

6.4 프로시저 없어 생기는 오류

View 에서 사용되는 함수가 지원지 않아 오류가 발생시, 이 함수는 SP 로 추정되는 경우 빈 SP 를 만들어 View 가 등록되도록 진행, 해당 SP 는 만들거나 응용에서 확인할 수 있도록 전달한다.

다음 예시는 View class load 시 함수가 없어 발생한 오류를 보여준다.

1 프로시저가 없어 오류 발생

```
csql> ;ru

In line 3, column 24,

ERROR: before ' ;
ALTER VIEW [testuser1].[example_status_alpha_view] ADD QUER...'
Function CUBE is undefined.
```

```
 1 ALTER VIEW [testuser1].[example_cube_view] ADD QUERY SELECT [year], status, COUNT(*) AS status_count
 2 FROM example_table_with_year
 3 GROUP BY CUBE([year], status);
 4 ALTER VIEW [testuser1].[example_status_alpha_view] ADD QUERY SELECT
 5     id,
 6     [year],
 7     status,
 8     REGEXP_SUBSTR(status, '[[:alpha:]]+') AS extracted_alpha
 9 FROM example_table_with_year;
10 ALTER VIEW [testuser1].[instr_v] ADD QUERY select instr(id,'a',1,1) AS id from test_table1;
11 ALTER VIEW [testuser1].[test_table1_calculated_view] ADD QUERY SELECT
12     id,
13     name,
14     get_value_by_name(name) AS calculated_value
15 FROM test_table1;
```

2 이관 시 임의로 함수 생성

*해당 함수는 기록이 필요하며, 담당자한테도 전달 필요

```
CREATE FUNCTION CUBE([year] int, status STRING) RETURN STRING
AS LANGUAGE JAVA
NAME 'CUBE(int,STRING) return STRING';
```

6.5 지원되지 않는 함수로 인한 오류

지원하지 않는 함수로 인한 오류 발생 시 CUBRID 에서 맞게 수정하거나 수정 자체도 불가능할 경우 해당 내역 정리하여 전달한다.

다음 예시는 View class load 시 함수의 차이점이 존재하여 발생한 오류를 보여준다.

1 INSTR 함수 오류 발생

함수 오류가 발생을 하였고 지원하는 함수인지 확인한다.

```
In line 16, column 46,

ERROR: before '  AS id from test_table1;
ALTER VIEW [testuser1].[test_table1_...'
Function instr is undefined.
```

```
ALTER VIEW [testuser1].[example_cube_view] ADD QUERY SELECT [year], status, COUNT(*) AS status_count
FROM example_table_with_year
GROUP BY CUBE([year], status);
ALTER VIEW [testuser1].[example_status_alpha_view] ADD QUERY SELECT
    id,
    [year],
    status,
    REGEXP_SUBSTR(status, '[[:alpha:]]+') AS extracted_alpha
FROM example_table_with_year;
ALTER VIEW [testuser1].[instr_v] ADD QUERY select instr(id,'a',1,1) AS id from test_table1;
ALTER VIEW [testuser1].[test_table1_calculated_view] ADD QUERY SELECT
    id,
    name,
    get_value_by_name(name) AS calculated_value
FROM test_table1;
```

2 INSTR 함수 구문 수정

오라클에서 사용하는 함수 INSTR()함수는 4 개의 Argument 를 지원하지만, CUBRID 는 3 개의 Argument 만 지원 우회 방안으로 구문 수정하여 입력한다.

- 구문 수정 전

```
select instr(id,'a',1,1) AS id from test_table1;
```

- 구문 수정 후

```
select instr(id,'a',(select instr(id,'a',1)+1)) AS id from test_table1;
```

```
ALTER VIEW [testuser1].[instr_v] ADD QUERY select instr(id,'a',1,1) AS id from test_table1;
ALTER VIEW [testuser1].[test_table1_calculated_view] ADD QUERY SELECT
    id,
    name,
    get_value_by_name(name) AS calculated_value
FROM test_table1;
```

6.6 Grant 파일 사용 방법

Grant 는 현 사용자가 부여받은 grant 에 대한 정보를 가지고 있다. 즉, grant 를 부여한 사용자의 스키마 정보에 만들어지지 않고, 권한을 가지고 있는 사용자의 스키마 정보중 grant 파일로 쓰여진다. 이관 대상이 아닌 경우 그 계정으로의 grant 는 의미가 없으며, 현 계정이 이관될 때 다른 스키마를 grant 받아 참고하고 있었다면 해당 테이블을 같이 이관할 필요가 있을 수도 있기 때문이다.

파일명은 다음과 같은 형태를 가지고 있다.

```
oradb_DB_USER2_grant.DB_USER3: 접두사는 기존 룰과 동일하다.
```

현재 DB_USER2 사용자가 grant 받은 정보에 대한 SQL 문장이며, grant 해준 사용자가 DB_USER3 인 것이다. 만약 DB_USER2, DB_USER3 를 모두 이관하는 경우 이 파일은 DB_USER3 계정에서 실행하여야 한다. DB_USER3 의 테이블에 대하여 DB_USER2 에게 grant 하는 것이기 때문이다.

아울러 synonym 이 있는 경우, 상대 계정에서 먼저 grant 를 해주어야 하므로, grant 를 먼저 상대 계정에서 수행 후 synonym 을 나중에 등록하여야 한다..

6.7 스키마 중복 오류

오라클 다른 2 개의 사용자를 CUBRID 사용자 하나로 이관 시 테이블 명이 겹칠 경우 오류 발생

이 경우 담당자와 협의하여 테이블 명을 변경하여 이관 후 응용의 수정이 필요하므로 변경 내역을 기록해 전달한다.

다음 예시는 오라클의 사용자 testuser1 과 testuser3 가 CUBRID testuser1 으로 이관 시 테이블 중복 오류를 보여준다.

1. Testuser3 스키마 중복 오류

   testuser1 의 스키마는 이미 이관된 상태이며, testuser3 의 스키마 입력 시 중복 오류가 나는 것을 확인할 수 있다.

```
csql> ;read TESTUSER3_testuser1_class

The file has been read into the command buffer.
csql> ;ru

In line 2, column 1,

ERROR: before ' (
[id] numeric(38,15),
[name] varchar(50),
[value] numeric(38...'
Class testuser1.test_table1 already exists.
```

2. Testuser3 스키마 중복 수정

   test_table1 테이블 명을 test_table1_testuser3 로 변경 진행한다.

```
CREATE TABLE [testuser1].[test_table1_testuser3](
[id] numeric(38,15),
[name] varchar(50),
[value] numeric(38,15)
)
```

   수정내역은 담당자에게 전달 필요하며, 추후에 사용하기 위해 수정한 파일을 별도로 저장한다.

```
csql> ;write TESTUSER3_testuser1_class_output

Command buffer has been saved.
```

3. 중복으로 인하여 테이블 명이 변경되면서 objects, pk, fk, view 파일 등 테이블 명 변경 필요

   변경 시 원본 파일은 건들지 말고 작업한다.

```
%class [testuser1].[test_table1_testuser3] ([id] [name] [value])
1. 'Name1' 2.
2. 'Name2' 4.
3. 'Name3' 6.
4. 'Name4' 8.
5. 'Name5' 10.
6. 'Name6' 12
```

```
ALTER /*+ NO_STATS */ TABLE [testuser1].[test_table1_testuser3] ADD CONSTRAINT [sys_c004030] PRIMARY KEY([id]);
ALTER /*+ NO_STATS */ TABLE [testuser1].[test_table3] ADD CONSTRAINT [sys_c004029] PRIMARY KEY([id]);
ALTER /*+ NO_STATS */ TABLE [testuser1].[test_table3_ref] ADD CONSTRAINT [sys_c004035] PRIMARY KEY([ref_id]);
```

7. 이관 전후 정상 이관 확인 방법

CMT 를 이용하여 데이터베이스를 이관하고 이관 전후의 데이터를 검증하기 위해선 export 한 데이터 건수와 import 한 데이터 건수를 확인하는 방법이 제일 무난하다. Export 과정에서 오류가 생겼으면 100% 진행이 안되며, import 과정에서 오류가 발생하였다면 export 한 건수와 import 한 건수가 다를 것이기 때문이다. 데이터의 세부사항에 대하여는 응용에서 최근 데이터와 일부 sampling 을 통해 정합성을 확인하여야 한다. 또는 업무 담당자가 원본과 이관 후 데이터베이스에 검색질의를 통해 결과의 비교가 필요할 수 있다.

앞서 이관 과정에서도 언급하였지만, CMT 이관 후 보고서의 요약 부분에 전체 레코드 건수가 표시되므로 이부분이 export 한 레코드 건수이고, CUBRID 데이터 loading 후 표시되는 총 데이터 건수가 import 된 건수 이므로 이 건수를 상호 비교하면 된다.

CMT 보고서의 record 항목 2002 과 Object load 화면의 Total 2002 object 동일하면 정상적으로 이관된 것이다.

- CMT 보고서 화면

시작 시간: 2023-12-04 15:27:24.382 종료 시간: 2023-12-04 15:27:28.436 총 소요 시간: 00 00:00:04.054

객체	원본에서 추출 ...	대상으로 입력 ...	실패	진행중
schema	0	0	0	100%
table	3	3	0	100%
view	0	0	0	100%
primary key	3	3	0	100%
foreign key	1	1	0	100%
index	0	0	0	100%
sequence	0	0	0	100%
synonym	0	0	0	100%
trigger	0	0	0	100%
function	0	0	0	100%
procedure	0	0	0	100%
grant	0	0	0	100%
record	2,002	2,002	0	100%

]

- Object load 시 화면

```
Start object loading.
testuser1.test_table3_ref 2 instances committed
testuser1.test_table3 1000 instances committed
testuser1.test_table1_testuser3 1000 instances committed
Total 2002 object(s) inserted, 0 object(s) failed.

*** Updating class statistics ***
Updated statistics for class testuser1.test_table1_testuser3.
Updated statistics for class testuser1.test_table3.
Updated statistics for class testuser1.test_table3_ref.

*** Closing the database ***
```

8. 스키마

8.1 예약어

예약어는 일부 상이하며, 사용시 “” , `(백틱, 작은따옴표가 아닌 ~키에 있는) 또는 [] 로 감싸서 사용하면 된다. 일반적으로 타입명, 함수명 등이 예약어가 된다. 자세한 예약어 리스트는 매뉴얼을 참고한다.

8.2 타입

대부분 유사하나, 명칭이 다르거나 지원되지 않는 부분에 대하여 사용 가능한 타입을 정리한다.

NULL 과 ‘’ 는 서로 다르게 취급하므로, not null default ‘’ 을 이용하거나 하여야 한다.

ORACLE	CUBRID	비고
bit boolean	BIT	BIT 로 사용
number	smallint, int, bigint, float/real, double, numeric	smallint:2byte int: 4byte bigint: 8byte numberic: 최대 38 자리(소수점포함) * unsigned 미지원
date	date time timestemp datetime	date: 날자 부분만 포함 time: 시간 부분만 포함 datetime: 날자, 시간, 1/1000 초 datetime: 날자, 시간 포함하며, 1/1000 초
lob	lob	외부에 파일로 저장. 내부에 저장하기 위해서는 clob→varchar, blob→bit varying 사용
char	char	고정길이이거나, 인덱스로 사용될 때 사용 고려
varchar, varchar2	varchar, string	string: varchar(1073741823)와 동일
long	varchar	CUBRID varchar 는 1G 까지 지원

8.3 제약조건

일반적인 제약조건들은 지원되며, 차이가 있는 부분들에 대하여 정리한다.

Key : PK, FK 모두 지원하며, key 로 지정 시 인덱스로도 같이 사용되므로 **별도로 인덱스를 생성할 필요 없다.**

Primary key : 모든 컬럼의 값이 not null 로 설정된다. 즉, 멀티 컬럼으로 PK 를 구성할 경우 어느 컬럼의 값도 null 이 허용되지 않는다.

Foreign key : on update 에 대하여 cascade 를 지원하지 않는다.

Check : 컬럼 속성 중 check 속성은 지원하지 않는다.

Constraint : key 이름 지정을 위한 constraint 는 컬럼 속성 뒤에 명시할 수 없고, 별도로 명시하여야 한다.

```
my_col int constraint pk primary key(my_col) : 지원안됨
my_col int,
constraint pk primary key(my_col)
```

제약조건을 비활성화 할 수 없다.

HA 구성인 경우 각 테이블의 PK 가 반드시 존재해야 합니다.

9. 질의

질의 사용시 사용법이 다르거나, 지원되지 않는 부분에 대하여 정리한다.

ORACLE	CUBRID	비고
SELECT SUM(COUNT(col_1)) FROM tbl_1 GROUP BY col_1	SELECT SUM(col_2) FROM (SELECT COUNT(col_1) AS col_2 FROM tbl_1 GROUP BY col_1)	집계함수 중복 사용 불가
from a, b where a.i = b.i(+)	from a left outer join b on a.i = b.i	outer join 시 ANSI 표준 사용 권장
from a full outer join b on a.i = b.i		full outer join 지원하지 않음
case when exists (select 1 from …)	case when 1=(select count(*) from …	case 문에서 exists 는 지원하지 않음.
order by col nulls first	9.2 부터 nulls first/last 지원	CUBRID 에서 null 은 제일 작은 값이다. 9.1 이하에서는 order by isnull(col), col

10. 연산자와 함수

10.1 연산자

ORACLE	CUBRID	비고
=	= <=>	<=>: null 과 비교시 is null 을 사용하지 않아도 가능
!= <> ^=	!= <>	
|| concat	|| concat +	문자열 합치기
minus	Difference	
Intersect	Intersection	
1/2 = 0.5	1/2 = 0	정수 연산시 둘다 정수이므로, 결과를 정수로 넘겨준다. oracle_compat_number_behavior 파라미터 사용 시 1/2=0.5 의 결과를 반환하나 의도치 않은 다른 변경이 발생할 수 있어 자세한 내용은 매뉴얼 참고한다.

datetime 연산시 숫자는 최소 단위인 1/1000 초로 계산

10.2 함수

함수는 종류가 많아 모두 다 열거하기는 어렵다. 일반적으로 많이 사용되는 함수를 기반으로 정리하였으며, 지원되는 모든 함수의 목록은 매뉴얼을 참고한다.

ORACLE	CUBRID	비고
sysdate	sysdate systime sysdatetime	사용되어 지는 데이터 타입에 맞게 사용
to_date	to_date to_time to_datetime	사용되어 지는 데이터 타입에 맞게 사용
instr	instr position	instr() 의 경우 3 개의 아규먼트만 지원한다.
dbms_random.value	rand drand	d 가 붙은 것은 0~1 사이의 실수를 넘겨준다.

	random drandom	rand() 는 질의 기반으로 난수를 만들어주며, random() 은 질의 결과 레코드 기반으로 난수를 만든다.
sys_guid		to_char(random(), '0000000000') + to_char(sys_datetime, 'YYYYMMDDHH24MISSFF) + to_char(random()%10000,'00000')
sys_extract_utc		지원하지 않음
xml 함수		지원하지 않음
over	over 함수	9.1 부터 지원되며, window 절은 현재 지원되지 않음.

10.3 일련번호

ORACLE	CUBRID	비고
sequence	serial	
	auto increment	데이터 입력시 자동증가 속성을 부여하면, 해당 컬럼의 값이 자동으로 주어진 값만큼 증가 create table my_tbl (id int auto_increment, name char(10)) insert into my_tbl(name) values('name1') → id 값은 자동 부여되며, 순차 증가

11. Stored Procedure(Procedure & Function)

Stored procedure 구현을 위해 사용되는 언어는 java 이다. 따라서 PG-SQL 구문을 java 구문으로 변경하여 사용한다.

배치 작업을 위한 procedure 에서의 사용은 무난하지만, function 등의 사용은 권장하지 않는다.

11.1 환경설정

아래 2 개의 환경변수가 설정되어 있어야 하며, 적용을 위해서는 CUBRID 서비스를 재구동해야 반영된다.

```
JAVA_HOME=/usr/java/jdk1.7.0_10
LD_LIBRARY_PATH=$JAVA_HOME/jre/lib/amd64:$JAVA_HOME/jre/lib/amd64/server
```

11.2 작성 방법

JDK1.8 이상(CUBRID 11.2 기준)

프로시저 내부에서 트랜잭션 명령(commit, rollback)을 사용할 수 없다. 사용하여도 무시되므로, 트랜잭션 처리는 프로시저 외부에서 하여야 한다.

11.2.1 프로시저/펑션 작성

java method 작성과 동일한 방법을 사용하면 된다. 굵은 글씨로 표시된 부분만 작성시 유의하면 된다. 프로스저/펑션으로 넘어오는 인자값과 리턴값에 대한 처리는 일반 java method 와 동일하게 하면 된다.

다음은 펑션 작성 예이다.

```
import java.sql.*;

public class mySP {
    public static void myfunc(int args) throws Exception {
        Connection conn = null;
        String ret_val = null;
        try {
            Class.forName("cubrid.jdbc.driver.CUBRIDDriver");
            conn = DriverManager.getConnection("jdbc:default:connection:");

            ...
            return ret_val;
        } catch ( SQLException e ) {
            System.err.println(e.getMessage());
        } catch ( Exception e ) {
            System.err.println(e.getMessage());
```

```
                } finally {
                    if ( conn != null ) conn.close();
                }
            }
        }
```

12. 성능

성능에 관련하여 지원되는 기능 및 질의 작성시 성능을 위해 고려할 부분도 정리한다.

12.1 힌트

12.1.1 조인힌트

ORACLE 과 동일한 형태로 사용하며, 조인 대상 한정이 가능하며, 지원되는 구문은 다음과 같다.

✶USE_IDX : 일반적인 netsted loop 조인시, 조인되어 지는 테이블의 조인 조건에 인덱스가 있는 경우 반드시 인덱스를 사용하여 조인을 한다.

✶USE_MERGE : sort merge join 을 사용한다.

✶USE_NL : nested loop join 을 사용한다.

✶ORDERED : from 절에 명시된 순서대로 join 을 수행한다.

12.1.2 인덱스힌트

ORACLE 과 달리 USING INDEX 라는 구문을 사용하며, 사용법은 다음과 같다.

```
Where …
USING INDEX a.idx1, b.idx2(+)
Order by
```

✶주어진 인덱스만 있는 것으로 간주하여, 주어진 인덱스와 풀 스캔중 비용이 저렴한 것을 선택한다.

✶조인시 특정 테이블에 대한 인덱스만 지정할 수 있으며, 이 경우 지정하지 않은 테이블은 옵티마이저가 선택한다.

✶주어진 인덱스 명이 잘못된 경우 오류를 발생한다.

✶(+) 를 이용하여 인덱스 사용 비용을 0 으로 만들 수 있다. 이는 해당 인덱스 사용을 강제하는 효과를 얻을 수 있다.

12.2 rownum

ORACLE 과 달리 rownum between 10 and 20 과 같은 형태로 사용할 수 있다. 다만 정렬전에 생성이 되어, 정렬과 함께 사용시 사용법이 좀 복잡해지므로 MySQL 형태의 limit 를 사용하면 훨씬 편리하다. 또한, inline view 를 사용하지 않아도 된다.

ORACLE

```
select rn, * from (
    select rownum rn, * from (
        select * from tbl where … order by …
        )
```

```
            where rownum <= 20
    )
    where rn >= 11
```

CUBRID

```
    select orderby_num() rn, * from tbl
    where … order by … limit 11, 20
```

✶정렬후 정렬된 순번을 얻고자 한다면, orderby_num() 을 사용하면 된다.

12.3 정렬

정렬을 하는 경우, 정렬을 위한 비용이 상당하므로 정렬을 피하기 위해 정렬 대상과 검색조건을 인덱스로 생성하게 되면, CUBRID 옵티마이저가 해당 인덱스를 사용하여 정렬을 회피할 수 있다. 이를 통해 검색 성능을 올릴 수 있다.

다만, 모든 경우에 대하여 성능 향상이 이루어 지지 않을 수 있으므로, 반드시 질의 수행 계획 및 성능 개선 여부를 확인하여야 한다.

```
    where id = 1 order by name desc

    create index idx1 on tbl(id, name desc)
```

12.4 max, min

max, min 대상을 인덱스로 생성시 인덱스를 이용하여 max, min 값을 얻을 수 있으므로 성능 향상을 꾀할 수 있다.

```
    Select max(id) from tbl
    Create index idx1 on tbl(id)
```

12.5 covering index

검색에 필요한 정보가 인덱스에 모두 존재하는 경우, 인덱스 정보만으로 데이터를 확인할 수 있으므로 성능 향상을 얻을 수 있다. 특히, 코드에 대한 이름을 얻는 형태의 조인의 경우 많은 도움이 될 수 있다.

```
    select name from tbl where id   = 1

    create index idx1 on tbl(id, name)
```

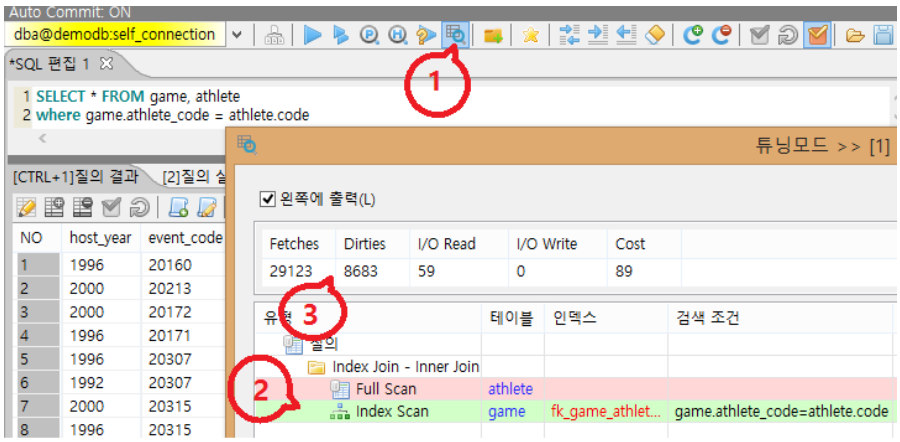
12.6 질의 수행 계획

질의 수행 계획을 보는 방법에 대하여 간단히 정리한다. 자세한 내용은 매뉴얼을 참고하기 바란다.

① 질의 수행 계획 및 통계 정보를 보기 위한 버튼.

② athelete table 에 대하여 full scan 을 하였으며, game table 에 대하여는 fk_game_athlete_code 를 이용하여 인덱스 스캔을 하였다는 의미

③ 질의 수행을 위해 발생한 I/O 양이 29123 페이지(CUBRID I/O 단위) 이며, 그중 최종 결과에는 사용되지 않는 페이지는 8683 페이지라는 의미. 결국 이 I/O 량을 줄이는 것이 제일 중요.



13. 응용 interface

응용 개발을 위한 driver 사용시 고려되어야할 부분을 정리한다.

13.1 연결 포트

CUBRID 응용에서는 데이터베이스와 연결을 할 때 반드시 포트를 지정해 주어야 한다. 기본적으로 제공되어 지는 포트는 **33000** 번이므로 이를 사용하면 된다.

만약 하나의 서버에 여러 개의 데이터베이스를 가지게 되면, **데이터베이스 마다 별개의 포트를 사용**하는 것을 권장한다.

CUBRID 는 3-tier 구조를 가지며, broker 라는 미들웨어에서 실제 데이터베이스와 연결하여 작업을 수행한다. broker 에서는 데이터베이스와의 연결에 대하여 pooling 기능을 가지고 있으므로, 이 기능을 제대로 활용하기 위해 broker 별로 데이터베이스 연결을 따로 가지도록 하는 것이다.

응용에서는 이 broker 와 연결을 하여 작업을 처리하게 되므로, 데이터베이스 별로 별개의 broker 를 구성하고 그 broker 에 주어진 포트를 사용하면 된다. broker 를 추가하는 방법은 별도로 매뉴얼을 참고하면 된다.

13.2 JDBC

JAVA 1.6 이상을 지원하며, 일반적은 작성 방법은 JDBC 표준을 따른다.

연결 문자열은 다음과 같으며, charset 을 지정해주어야 문자셋이 깨지지 않고 저장될 수 있다.

HA 환경에서는 url_ha 형태와 같이, **althosts** 를 stand-by 서버로 지정해주어야 한다.

```
import java.sql.*;
import cubrid.jdbc.driver.*;
String url = "jdbc:cubrid:192.168.0.100:33000:demodb:::?charset=utf8";
String url_ha = "jdbc:cubrid:192.168.0.100:33000:demodb:::?
althosts=192.168.0.101:33000& charset=utf8";

Class.forName("cubrid.jdbc.driver.CUBRIDDriver");
Connection conn = DriverManager.getConnection(url, "db_user", "dbpw");
```

13.3 PHP

PHP 의 경우 함수에 대한 표준이 없다. 다만 대부분의 함수들이 prefix 부분을 제외하고는 유사하므로, 기존 사용함수의 prefix 부분만 cubrid 로 변경하여 검색해보면 된다.

HA 를 사용할 경우, url 형식의 연결 문자열(JDBC 참고)을 사용하는 cubrid_connect_with_url 이라는 함수를 사용하여야 하며, 사용방법은 다음과 같다. HA 를 사용하지 않더라도 해당 함수는 사용 가능(althosts 부분만 제거)하다.

```
        $url_ha                 =                 “jdbc:cubrid:192.168.0.100:33000:demodb:::?
althosts=192.168.0.101:33000& charset=utf8”;

        $con = cubrid_connect_with_url(url, “db_user”, “dbpw”);
```

기본적으로 자동 커밋모드로 동작한다. 자동 커밋을 끄기 위해서는
cobrid_set_autocommit($con, CUBRID_AUTOCOMMIT_FALSE) 를 이용하거나,
연결문자열 뒷부분에 autocommit=false 를 추가해주면 된다.

14. HA 구성시 고려사항

장애 대비를 위해 자체적으로 지원되고 있는 HA 를 사용시 주의할 사항을 정리한다.

주의사항	비고
OS	LINUX 에서만 사용가능하다.

Primary key	스토리지 장애에도 대처하기위해, 스토리지를 공유하지 않은 방식을 사용한다. 따라서 서로 다른 별개의 데이터베이스를 복제를 통해 데이터 동기를 유지하므로, 반드시 PK 가 존재하여야 한다.
	다음은 기본키가 없는 테이블을 찾는 질의이다.
	`select owner_name, class_name`
	`from db_class`
	`where class_type = 'CLASS' and is_system_class = 'NO'`
	`and class_name not in`
	`( select class_name from db_index where is_primary_key ='YES' )`
	`order by owner_name, class_name`

trigger	9.2 부터 사용 가능

lob	lob 데이터가 별도의 파일로 저장되며, 이 파일이 복제되지 않으므로 사용할 수 없으며, bit varying 을 사용하여야 한다.
	다음은 lob 을 사용하는 테이블과 컬럼을 찾는 질의이다.
	`select class_name, attr_name, data_type`
	`from db_attribute`
	`where data_type like '%LOB'`
	`order by class_name, attr_name`