What is CUBRID Transaction Capture API?

CUBRID Transaction Capture 기능(이후 CTC 로 표기)은 source system 에 존재하는 CUBRID 데이터베이스 서버(이후 srcDB 로 표기)의 특정 테이블에 수행되는 트랜잭션에 의한 변경사항을 원격 타겟 시스템(이후 trgSys 로 표기)에 적용하는데 필요한 트랜잭션 정보를 제공한다. 이를 위해 srcDB 와 동일한 노드에 CTC 프로세스가 CTC 서버로 운영되며, CUBRID Transaction Capture API(이후 CTC API 로 표기)는 CTC 프로세스와 네트워크 통신을 하는 어플리케이션에서 CTC 기능을 이용할 수 있도록 제공하는 인터페이스들의 집합이다.

CTC API

ARCHITECTURE 와 API 방식과 관련하여

⎫ **NO_DAEMON_LIBRARY_API**

이 방식은 CTC 기능을 제공하기 위해 어떤 서버 프로세스도 뜨지 않는다. 단순히 기능을 수행할 수 있는 정도의 API 만 제공하는 방식으로 필요하다면 사용자가 알아서 daemon 을 만들어 써야 한다. 일단 구현의 복잡성이 가장 낮고 프로젝트 기간동안 구현의 완료 후 많은 테스트를 해볼 수 있기 때문에 라이브러리 자체의 안정성은 꽤 확보할 수 있을 것으로 보인다.
그러나 과연 이런 방식의 API 를 제공한다면 사용성이 있을 것인지 의문이다. 왜냐하면 원격에서 그것도 필요할 때마다 API 를 사용해서 원격의 log 파일에 접근해 읽어오는 것이 가능하다 하더라도 트랜잭션내 하나의 오퍼레이션을 수행하는데 network 비용과 disk I/O 가 모두 개별적으로 수반되게 된다. 그냥 시험삼아 CTC 기능을 사용해 보거나 매우 드물게 트랜잭션이 발생하는 테이블에 대해서만 제한적으로 이용할 수 있을 것이다. 또한 이런 라이브러리 방식의 특성상 확장성이 떨어지게 되어 추후 CTC 기능의 업데이트가 용이하지 않을 것으로 생각된다.

⎫ **CTC_DAEMON_LIBRARY_API**

이 방식은 CTC 어플리케이션을 구동하는 시스템에 CTC daemon 프로세스를 띄우는 방식이다. 따라서 CTC daemon 프로세스를 구동하기 위한 API 부터 제공해야 한다. 그러나 문제는 어플리케이션을 기동하는 시스템 환경에 대해 전혀 알 수가 없기 때문에 CTC daemon 프로세스가 정상적으로 기동하여 동작하기 위해서는 굉장히 많은 부분들을 고려하여 설계해야 하고 반대로 CTC 프로세스를 위한 시스템 제약사항을 두는 경우 사용자 어플리케이션 자체가 CTC 프로세스의 환경에 종속적이게 되어 다양한 시스템에서 다양한 방식으로 어플리케이션을 만들어 사용할 수 없게 된다.

⎫ **CTC_SERVER_ACCESS_API**

CTC 서버 프로세스를 source system 인 CUBRID database server 와 동일한 노드에서 기동하고 제공된 API 를 통해 CTC 서버에 필요한 오퍼레이션을 수행하는 방식으로 가장 무난한 방식이긴 하나 어플리케이션과의 통신을 위한 프로토콜 설계가 필요하다. 또한 아래와 같은 것들을 고려해야 한다.
1. CTC 프로세스를 클라이언트 별로 띄울 필요가 있는가? 아니 다수의 클라이언트를 고려할 필요가 있는가? 필요가 있다면, 소스 시스템에 여러 CTC 프로세스가 뜰 경우 프로세스 수의 제한은 무엇을 기준으로 할 것인가? 또한 프로세스간 간섭이 발생할 여지는 없는가?
2. CTC 기능의 사용을 위한 권한의 제약을 둘 것인가? 권한을 둔다면 권한의 검사는 API 를 통해 수행할 것인가? 그렇다면 CTC 프로세스가 소스 데이터베이스에 접속해야 한다. 아예 소스 데이터베이스에 접속하여 정보들을 가져오는 방식으로 설계하는게 나을까? API 로 주지 않는다면 CTC 는 API 를 수행하는 어플리케이션이 이미 권한을 갖고 있다는 가정을 해야 한다. 그렇다면 과연 그 가정은 항상 타당한가? 그렇지 않다. API 는 오픈이므로 어플리케이션 개발자가 악의적으로 권한검사를 누락한다면?
3. CTC 보안을 위한 장치가 필요한가?

CTC API table
아래 테이블은 API 가 추가되거나 변경되는 경우 필요에 따라 수정될 예정

#	CATEGORY	API	DESCRIPTION
1	C	CTC_OPEN_CONNECTION	CTC 서버와 connection 을 생성한다
2	C	CTC_CLOSE_CONNECTION	CTC 서버와의 connection 을 종료한다
3	J	CTC_ADD_JOB	지정한 connection 에 job 을 하나 추가한다
4	J	CTC_DELETE_JOB	지정한 connection 에서 해당 job 을 제거한다
5	I	CTC_CHECK_SERVER_STATUS	CTC 서버 프로세스의 상태 정보를 확인한다
6	J	CTC_REGISTER_TABLE	특정 JOB 에 복제할 테이블을 등록한다
7	J	CTC_UNREGISTER_TABLE	특정 JOB 으로부터 등록된 테이블을 제거한다
8	J	CTC_START_CAPTURE	특정 JOB 에 대해 트랜잭션 capture 를 시작한다
9	J	CTC_STOP_CAPTURE	특정 JOB 의 트랜잭션 capture 를 종료한다
10	J	CTC_FETCH_CAPTURE_TRANSACTION	capture 한 트랜잭션을 fetch 한다
11	I	CTC_CHECK_JOB_STATUS	해당 job 에 대해 fetch 할 결과물이 있는지 CTC 서버에 확인한다
12	J	CTC_SET_JOB_ATTRIBUTE	queue size 와 같은 해당 job 의 속성값을 설정한다
13	J	CTC_GET_STATISTICS	추출된 로그의 개수와 같은 통계 정보를 얻어온다

- **API Category**
  API Category 는 CTC API 를 사용하는 어플리케이션 관점에서 API 를 유형별로 분류하기 위한 것으로 아래와 같다.

Category	Notation	Description

CONNECTION	C	네트워크 연결과 관련된 API
INFO	I	상태정보나 통계정보, 특정 JOB 에 대한 정보 등과 관련된 API
JOB	J	JOB 의 생성 및 구성, 시작과 종료, 결과의 fetch 와 관련된 API

1. CTC_OPEN_CONNECTION

description:

CTC 서버 프로세스와 통신하기 위한 세션을 생성하고 CTC 프로세스와 통신을 시도한다. CTC 서버 프로세스는 어플리케이션에서 이 API 를 수행한 뒤에라야 어플리케이션을 최초로 identify 하게 된다. 기본적으로 control session 과 job session 을 각각 하나씩 생성하여 connection 을 맺지만 connection_type 입력 파라미터의 설정값에 따라 control session 만 생성할 수도 있다.

return value (type):

int ctc_handle

- 해당 connection 에 대한 핸들. 어플리케이션과 CTC 서버간의 연결과 관련하여 보다 자세한 내용은 CUBRID Transaction Capture Concept Design 을 참조) 어플리케이션은 이후 mapping 된 dedicated communication channel 을 통해 CTC 프로세스와 통신하게 된다.

input parameters:

int connection_type

- open 하려는 connection 의 타입, CTC 에는 control session 과 job session 이 있다. control session 의 경우 상태정보나 CTC 프로세스를 control 하기 위한 request 를 전달하고 응답받기 위한 session 이고 job session 은 실제 job 수행을 통해 요청한 data 를 전달받기 위한 세션이다. control session 은 job session 은 CTC_ADD_JOB API 가 수행될 때마다 추가되며 CTC_DELETE_JOB API 를 통해 제거할 수 있다. connection_type 은 아래 표와 같이 CTC_CONN_TYPE_CTRL_ONLY 와 CTC_CONN_TYPE_DEFAULT 두가지가 있다.

CTC_CONN_TYPE	VALUE	DESCRIPTION
CTC_CONN_TYPE_DEFAULT	0	control session 과 하나의 job session 을 생성
CTC_CONN_TYPE_CTRL_ONLY	1	control session 만 생성

CTC_CONN_TYPE_CTRL_ONLY 의 용도

모니터링 용도로만 사용할 어플리케이션을 개발하는 경우 job session 은 어플리케이션, CTC 프로세스 모두에 사실상 필요없는 자원의 낭비가 되므로 control session 만을 유지하도록 하기 위함

char * connection_string

- CTC 서버에 대한 접속정보 (필요에 따라 옵션의 추가가 가능)

connection info 를 입력받는 방식과 관련하여

크게는 아래와 같은 3 가지 방식을 생각해볼 수 있을 듯 하다. 어떤 방식으로 처리할 것인지는 선택의 문제가 될 것이나 이번 프로젝트에서는 connection string 을 입력받는 방식이 가장 나아보인다.

- **connection string**
  JAVA 의 connection string 같은 문자열로 접속정보를 입력받아 라이브러리에서 처리해주는 방식
  문자열에 대한 가독성이 떨어지긴 하나 사용이 간편하고 옵션의 추가나 변경사항의 적용 등이 상대적으로 간편하다.
- **structure**
  C style 의 접속 addr 구조체를 naive 하게 입력받는 방식
  메뉴얼과는 별도로 헤더파일을 제공하거나 이를 위한 API 를 추가해줘야 한다.
- **attributes**
  함수의 input parameter 로 접속 정보들을 하나하나 열거해서 넘겨주는 방식
  단순하나 차후 변경사항이나 옵션의 추가와 같은 상황이 발생하는 경우 내부코드의 수정은 물론 API 자체의 수정이 필요하다.

2. CTC_CLOSE_CONNECTION

description:

CTC 서버 프로세스는 이 API 를 수행한 어플리케이션의 ctc_handle 로 관리되는 모든 session 에 대해 연결을 종료하고 최종적으로 해당 핸들을 제거한다. CTC 라이브러리 또한 CTC 서버로부터 반환되는 결과값에 따라 자신이 갖고 있던 해당 handle 과 관련된 모든 자료구조를 메모리에서 정리한다.

return value:

- 기본적으로 SUCCESS / FAIL
- CTC 서버는 CTCP 를 통해 아래와 같은 수행결과를 어플리케이션에 전달하고 어플리케이션은 CTC_SUCCESS 와 CTC_FAILURE 를 구분하여 처리

RETURN	DESCRIPTION
CTC_RC_SUCCESS	dedicated session 을 정상적으로 종료하고 free 한 경우의 반환 값
CTC_RC_FAILED_INVALID_HANDLE	해당 핸들이 CTC 프로세스 내에 존재하지 않거나 invalid 한 핸들인 경우
CTC_RC_FAILED_SESSION_IS_BUSY	session 에 할당된 job 의 수행으로 인해 특정 시간동안 retry 후에도 session 의 종료처리를 수행하지 못한 경우
CTC_RC_FAILED_CLOSE_SESSION	위 2 가지 경우를 제외한 session close 실패 시

input parameters:

int ctc_handle

- 종료하고자 하는 connection 에 대한 핸들로 CTC_OPEN_CONNECTION 을 통해 CTC 프로세스로부터 전달받은 값이다.

int close_condition

- 수행중인 job 에 대한 처리 조건

CONDITION	VALUE	DESCRIPTION
CTC_QUIT_JOB_IMMEDIATELY	0 (default)	해당 job 을 바로 종료한다
CTC_QUIT_JOB_AFTER_TRANSACTION	1	CTC 서버에서 어플리케이션으로 전송 대기중인 captured 트랜잭션을 모두 전송하고 job 을 종료한다

3. CTC_ADD_JOB

description:

job 을 하나 추가한다.

return value (type):

job_descriptor (integer)

- 이 API 의 수행으로 JOB 리스트에 추가된 JOB 의 descriptor, 실패한 경우 -1 을 반환

input parameters:

int ctc_handle

4. CTC_DELETE_JOB

description:

특정 job 을 제거한다.

return value (type):

- 기본적으로 SUCCESS / FAIL
- CTC 서버는 CTCP 를 통해 아래와 같은 수행결과를 어플리케이션에 전달하고 어플리케이션은 CTC_SUCCESS 와 CTC_FAILURE 를 구분하여 처리

RETURN	DESCRIPTION

CTC_RC_SUCCESS	job 의 제거가 성공
CTC_RC_FAILED_SESSION_NOT_EXIST	해당 세션이 CTC 프로세스 내에 존재하지 않는 경우로 SUCCESS 처리
CTC_RC_FAILED_INVALID_HANDLE	해당 핸들이 CTC 프로세스 내에 존재하지 않거나 invalid 한 핸들인 경우
CTC_RC_FAILED_SESSION_CLOSE	해당 job 에 대한 session close 실패 시
CTC_RC_FAILED_DELETE_JOB	해당 JOB 을 제거하는데 실패한 경우

FAILURE 시의 error number 와 관련하여

(i)아래의 카테고리에 포함되지 않는 모든 에러는 라이브러리 수행 과정에서 발생하는 일반적인 에러로 보아 -999 ~ -1 사이의 error number 를 할당하였다.
따라서 CTCP 를 통해 CTC 서버로부터 전달받는 에러들은 모두 -1000 이후, 즉 아래의 카테고리 내에서 구분된다.

CATEGORY	#
CTC_PROCESS	-1100
CTC_HANDLE	-1200
SESSION	-1300
JOB	-1400
TABLE	-1500

input parameters:

int ctc_handle

int job_descriptor

- 제거하고자 하는 job

5. CTC_CHECK_SERVER_STATUS

description:

CTC 는 CTC 서버 프로세스가 정상적으로 기동 중인 경우에만 사용할 수 있는 기능이므로 어플리케이션은 CTC 서버의 상태를 먼저 확인해야 한다. 다른 API 들 모두 내부적으로 가장 먼저 CTC 서버의 상태를 확인하고 진행한다. (API 의 수행 목적과 기능적인 면을 생각해보았을 때 CTC 서버의 상태확인에 따른 네트워크 비용이 성능에 미치는 영향은 거의 없다)
(/)CTC 프로세스로부터 전달받은 서버의 상태정보가 CTC_SERVER_RUNNING 인 경우 SUCCESS, 그렇지 않으면 FAILURE 를 반환할지 상태정보 자체를 반환할지는 미정

return value:

- 기본적으로 +return 은 SUCCESS / -return 은 FAIL
- CTC 서버는 CTCP 를 통해 서버의 상태정보를 어플리케이션에 전달하고 어플리케이션은 해당 return 값을 구분하여 처리

input parameters:

int ctc_handle

SUCCESS 시 CTCP 로 전달받는 CTC 서버의 상태 정보

- CTC 서버로부터의 reply packet 에 아래 상태정보들 중 하나가 셋팅된다.

CTC_SERVER_STATUS	VALUE	DESCRIPTION
CTC_SERVER_NOT_READY	0	CTC 서버가 job 을 처리 가능한 상태가 아닌 경우
CTC_SERVER_RUNNING	1	CTC 서버가 정상적으로 동작중인 경우의 상태
CTC_SERVER_CLOSING	2	CTC 서버 프로세스를 종료중인 경우의 상태

CTC process info 와 관련하여

관리 목적의 보다 다양한 정보들을 유의미하게 분류하여 정리한 뒤 차 후 아래와 같은 별도의 API 를 통해 제공할 수 있다.
단, 특정 API 들에 대해서는 수행권한 등에 대한 고려가 필요할 것으로 보인다. (본 프로젝트에서는 기간의 제약상 범위에서 제외)

- CTC_GLOBAL_JOB_INFO
- CTC_GLOBAL_CONNECTION_INFO

6. CTC_REGISTER_TABLE

description:

특정 JOB 에 테이블을 등록한다. 어플리케이션 개발자는 이 API 를 수행하여 복제하고자 하는 테이블을 CTC 서버에 등록할 수 있다.

return value (type):

- 기본적으로 SUCCESS / FAIL
- CTC 서버는 CTCP 를 통해 아래와 같은 result code 를 어플리케이션에 전달하고 어플리케이션은 CTC_SUCCESS 와 CTC_FAILURE 를 구분하여 처리

RESULT CODE	DESCRIPTION
CTC_RC_SUCCESS	해당 job 에 테이블 등록 성공

CTC_RC_FAILED_INVALID_HANDLE	해당 핸들이 CTC 프로세스 내에 존재하지 않거나 invalid 한 핸들인 경우
CTC_RC_FAILED_INVALID_JOB	해당 JOB 이 CTC 프로세스 내에 존재하지 않거나 invalid 한 경우
CTC_RC_FAILED_TABLE_REGISTER	테이블 등록이 실패한 경우

input parameters:

int ctc_handle
int job_descriptor

- 테이블을 등록하려는 job

  char * db_user_name

- 테이블의 사용자 명

  char * table_name

- 등록하려는 테이블 명

7. CTC_UNREGISTER_TABLE

description:

CTC 프로세스에 등록한 테이블을 복제대상에서 제거한다. 파라미터로 입력받은 table 을 찾아 해당 job 의 복제대상 list 에서 제거한다.

return value (type):

- 기본적으로 SUCCESS / FAIL
- CTC 서버는 CTCP 를 통해 아래와 같은 result code 를 어플리케이션에 전달하고 어플리케이션은 CTC_SUCCESS 와 CTC_FAILURE 를 구분하여 처리

RESULT CODE	DESCRIPTION
CTC_RC_SUCCESS	해당 job 의 복제대상 list 로부터 테이블 제거 성공
CTC_RC_FAILED_UNREGISTERED_TABLE	job 에 해당 테이블이 등록되어 있지 않은 경우, (어플리케이션 입장에서는 성공처리)
CTC_RC_FAILED_INVALID_HANDLE	해당 핸들이 CTC 프로세스 내에 존재하지 않거나 invalid 한 핸들인 경우

CTC_RC_FAILED_INVALID_JOB	해당 JOB 이 CTC 프로세스 내에 존재하지 않거나 invalid 한 경우
CTC_RC_FAILED_TABLE_UNREGISTER	테이블 unregister 자체가 실패한 경우

input parameters:

int ctc_handle

int job_descriptor

- 제거하려는 테이블을 list 에 갖고 있는 job

  char * db_user_name

- 테이블의 사용자 명

  char * table_name

- 등록하려는 테이블 명

8. CTC_START_CAPTURE

description:

이 API 를 통해 등록한 테이블들에 수행되는 트랜잭션을 capture 할 수 있다. CTC 는 어플리케이션에서 등록한 일련의 테이블들에 수행되는 트랜잭션들에 대한 capture 를 하나의 job 으로 인식한다. 그리고 CTC_START_CAPTURE 의 수행을 통해 이러한 job 의 수행 및 접근 권한을 획득할 수 있다. 따라서 CTC_REGISTER_TABLE API 를 통해 테이블을 등록했다 하더라도 이 API 를 수행하지 않으면 트랜잭션의 복제는 수행되지 않는다. 이 API 를 수행하면 CTC 프로세스로부터 결과값을 가져오며 job descriptor 는 CTC_STOP_CAPTURE API 를 수행하면 invalidate 된다.

return value (type):

- 기본적으로 SUCCESS / FAIL
- CTC 서버는 CTCP 를 통해 아래와 같은 result code 를 어플리케이션에 전달하고 어플리케이션은 CTC_SUCCESS 와 CTC_FAILURE 를 구분하여 처리

RESULT CODE	DESCRIPTION
CTC_RC_SUCCESS	성공
CTC_RC_FAILED_INVALID_HANDLE	해당 핸들이 CTC 프로세스 내에 존재하지 않거나 invalid 한 핸들인 경우
CTC_RC_FAILED_INVALID_JOB	해당 JOB 이 CTC 프로세스 내에 존재하지 않거나 invalid 한 경우

CTC_RC_FAILED_START_CAPTURE	capture start 수행 자체에 실패한 경우

input parameter:

int ctc_handle

int job_descriptor

- 트랜잭션을 capture 하려는 job 의 id

9. CTC_STOP_CAPTURE

description:

이 API 는 CTC 서버에서 수행되고 있는 특정 job 을 중지하기 위해 수행한다.
capture 를 restart 할 수 있는지와 관련하여

review 를 통해 결정된 CTC_STOP_CAPTURE 의 기능범위는 job 의 capture 동작을 중지하는 것까지이다.
따라서 사용자는 해당 job 에 대해 반복적으로 CTC_START_CAPTURE 와 CTC_STOP_CAPTURE 를 수행할 수 있다.
CTC_STOP_CAPTURE 후 CTC_DELETE_JOB 을 통해 해당 job 을 제거하지 않고 restart 하게 되는 경우 즉, API 의 반복 사용으로 인해 data consistency 를 보장할 수 없는 상황에 대한 책임은 어플리케이션에 있다는 것을 매뉴얼이나 가이드 문서 작성시 기술해야 한다.

return value:

- 기본적으로 SUCCESS / FAIL
- CTC 서버는 CTCP 를 통해 아래와 같은 result code 를 어플리케이션에 전달하고 어플리케이션은 CTC_SUCCESS 와 CTC_FAILURE 를 구분하여 처리

RESULT CODE	DESCRIPTION
CTC_RC_SUCCESS	성공
CTC_RC_FAILED_INVALID_HANDLE	해당 핸들이 CTC 프로세스 내에 존재하지 않거나 invalid 한 핸들인 경우
CTC_RC_FAILED_INVALID_JOB	해당 JOB 이 CTC 프로세스 내에 존재하지 않거나 invalid 한 경우
CTC_RC_FAILED_STOP_CAPTURE	capture stop 수행 자체에 실패한 경우

input parameter:

int ctc_handle

int job_descriptor

- capture 수행을 중지하려는 job

10. CTC_FETCH_CAPTURED_TRANSACTION

description:

CTC_START_CAPTURE 를 수행하게 되면 CTC 프로세스는 입력받은 job 에 대하여 트랜잭션 capture 를 시작한다. 이 때 어플리케이션에서는 이 API 를 수행함으로써 capture 된 트랜잭션 정보를 fetch 해올 수 있다.
이 API 의 결과값은 트랜잭션을 구성하는 하나의 오퍼레이션이다.

return value (type):

int result_op_str_length

- result_buffer 에 담은 json item 형식의 오퍼레이션 문자열의 길이, -1 인 경우 buffer size 가 결과 문자열의 size 보다 작아 FAIL 이라는 것을 의미

input parameters:

int ctc_handle

int job_descriptor

- capture 한 트랜잭션 결과를 fetch 할 대상 job

  char * result_buffer

- 어플리케이션에서 할당한 버퍼로 API 라이브러리로부터 결과물을 전달받기 위한 buffer

  int result_buffer_size

- result_buffer 의 크기

  int * required_buffer_size (OUT)

- 실제 필요한 버퍼 크기 (버퍼크기가 실제 데이터보다 작아서 fail 하는 경우 out parameter 로 값이 설정된다)

11. CTC_CHECK_JOB_STATUS

description:

CTC_START_CAPTURE 를 수행한 어플리케이션은 복제하고자 하는 테이블에 트랜잭션이 수행되는 시점을 알 수 없고 commit 되어 fetch 가능한 상태가 되는 시점도 알 수 없다.
따라서 CTC 서버로부터 해당 job 의 상태정보를 얻어와야 한다. 상태정보에 따라 어플리케이션은 CTC_FETCH_CAPTURED_TRANSACTION 을 수행하거나 CTC_STOP_CAPTURE 를 통해 해당 job 을 제거할 수도 있다.

return value (type):

- 기본적으로 SUCCESS / FAIL
- CTC 서버는 CTCP 를 통해 아래와 같은 result code 결과를 어플리케이션에 전달하고 어플리케이션은 CTC_SUCCESS 와 CTC_FAILURE 를 구분하여 처리

RESULT CODE	DESCRIPTION
CTC_RC_SUCCESS	성공
CTC_RC_FAILED_INVALID_HANDLE	해당 핸들이 CTC 프로세스 내에 존재하지 않거나 invalid 한 핸들인 경우
CTC_RC_FAILED_INVALID_JOB	해당 JOB 이 CTC 프로세스 내에 존재하지 않거나 invalid 한 경우
CTC_RC_FAILED_GET_JOB_STATUS	해당 job 의 상태정보를 얻어오는데 실패한 경우

input parameters:

int ctc_handle

int job_descriptor

SUCCESS 시 reply packet 에 설정되는 job 의 상태 정보들

CTC 서버로부터의 reply packet 에 아래 상태정보들 중 하나가 셋팅된다.

CTC_JOB_STATUS	VALUE	DESCRIPTION
CTC_JOB_NONE	0	job description 에 해당하는 job 이 존재하지 않는 상태
CTC_JOB_WAITING	1	job 에 수행되고 있는 트랜잭션이 존재하지 않는 상태
CTC_JOB_PROCESSING	2	job 에 트랜잭션이 수행되고 있는 상태
CTC_JOB_READY_TO_FETCH	3	트랜잭션 수행 후 fetch 할 결과물이 대기중인 상태
CTC_JOB_CLOSING	4	job description 에 해당하는 job 을 closing 하고 있는 상태

CTC 서버로부터 job 의 상태정보를 얻어가는 방식과 관련하여

방식은 크게 두가지를 생각해볼 수 있다. 첫번째는 CTC 서버에서 어플리케이션으로 알람을 주는 방식이고 두번째는 주기적으로 어플리케이션이 CTC 서버에 해당 정보를 요청하는 방식이다.
첫번째 방식의 경우 결과물에 대한 적시성이란 관점에서 선택할 수 있는 방식이나 구현이 다소 복잡해진다.
두번째 방식의 경우 어플리케이션이 해당 job 에 대한 상태정보를 명시적으로 얻을 수 있으며 구현이 간단하다.
이번 프로젝트에서는 두번째 방식으로 설계 구현한다.

12. CTC_SET_JOB_ATTRIBUTE

description:

어플리케이션은 CTC_ADD_JOB 을 통해 생성한 JOB 에 대해 필요한 속성값을 설정할 수 있다.
JOB 에 설정가능한 속성값과 관련하여

지정한 테이블들에 대해 수행되는 트랜잭션의 capture 가 CTC 의 주된 기능이며 해당 작업의 구분이 바로 JOB 이라는 단위이다.
현재는 이러한 JOB 과 관련하여 설정가능한 속성이 queue size 정도이나 이 후 기능의 추가나 구조의 변경이 이루어지는 경우를 위해 속성값을 별도로 설정할 수 있는 API 및 CTCP 오퍼레이션을 추가한다.

return value (type):

- 기본적으로 SUCCESS / FAIL
- CTC 서버는 CTCP 를 통해 아래와 같은 result code 결과를 어플리케이션에 전달하고 어플리케이션은 CTC_SUCCESS 와 CTC_FAILURE 를 구분하여 처리

input parameters:

int ctc_handle

int job_descriptor

- 속성값을 설정할 job

  int job_attr_id

- 설정할 속성의 id 로 아래와 같다.

  JOB ATTRIBUTES

ID	JOB ATTRIBUTE	DESCRIPTION
1	CTC_JOB_QUEUE_SIZE	job 에 할당되는 트랜잭션 전송 큐의 크기
2	CTC_LONG_TRAN_QUEUE_SIZE	job 에 할당되는 long term transaction 전용 대기 큐의 크기

13. CTC_GET_STATISTICS

description:

어플리케이션이 CTC 기능의 수행을 통해 얻을 수 있는 각종 통계정보를 얻는데 사용할 수 있다.

return value (type):

- 기본적으로 SUCCESS / FAIL

input parameters:

int ctc_handle

int job_descriptor

int stat_id

STATISTICS_ID	VALUE
CTC_SRV_EXTRACTED_LOG_COUNT	1
CTC_FILTERED_LOG_COUNT	2

output parameter

int stat_value