

## 1. CUBRID 2008 R3.0 변경 정보

- 본 문서는 릴리스노트를 바탕으로 추가된 기능들에 대하여 설명 및 예제를 추가하여 이해를 도울 수 있도록 가이드 한다.

### CUBRID 2008 R3.0 특징

CUBRID 2008 R3.0 릴리스는 CUBRID 기반의 응용 프로그램을 보다 쉽고 편리하게 작성할 수 있도록 다양한 연산자, 함수 및 구문을 확장하였다. 또한, 서비스 운영 편의성을 향상시키기 위하여 **cubrid compactdb** 유틸리티를 개선하고 CUBRID HA 기능 안정화에 주력하였다.

- [SQL 문법 확장 지원](#)

- **CREATE** 문 확장: 기존 테이블의 스키마 복제, 데이터 전체 복제를 통한 테이블 생성 기능 지원
- **INSERT** 문 확장: 멀티 레코드 입력 가능, **REPLACE** 문 지원
- **SELECT** 문 확장: **LIMIT** 절 지원, **GROUP BY** 절 문법 확장 지원
- **ALTER** 문 확장: 추가할 컬럼의 위치 지정 기능, 인덱스 및 제약 조건 추가/삭제 기능 확장 지원
- 그 외, **TRUNCATE** 문, prepared statement 문 등을 지원

- [연산자 및 함수 추가 지원](#)

- 논리 연산자, 비교 연산자, 비트 연산자 지원
- 다양한 수학 함수, 날짜/시간 데이터 함수 및 날짜 출력 포맷 지정자 추가 지원
- 다양한 문자열 함수를 추가 지원
- PHP API에 20여 개 함수 추가 지원

- [운영 편의성 개선](#)

- CUBRID HA 기능 안정화
- 데이터베이스 공간 정리 작업을 온라인에서 수행할 수 있도록 **cubrid compactdb** 유틸리티에 옵션 추가 지원

보다 자세한 내용은 아래의 [CUBRID 2008 R3.0에서 변경된 사항](#)을 참고한다.

### CUBRID 2008 R3.0 버전으로 업그레이드하는 방법

#### 업그레이드 주의 사항

- 추가된 예약어 및 허용되지 않는 식별자 확인

SQL 구문 확장 및 함수/연산자 추가 지원으로 인해 아래의 예약어가 추가되었으며, 식별자에 포함할 수 없는 특수 문자가 변경되었다. 이전 버전에서 이러한 식별자가 사용된 경우라면, 업그레이드 이후 오류가 발생하게 되므로 주의하여야 한다.

업그레이드 이후 응용에서 추가된 예약어 사용에 주의를 해야 하므로, 아래 마이그레이션 절차와 같이 진행을 해야 한다.

보다 자세한 사항은 [데이터베이스 마이그레이션 절차](#), [SQL 설명서> 예약어](#) 및 [SQL 설명서> 식별자](#)를 참고한다.

항목	CUBRID 2008 R3.0에서 허용하지 않는 식별자
추가된 예약어	DATABASE, DAY_MILLISECOND, DAY_SECOND, DAY_MINUTE, DAY_HOUR,

	DISTINCTROW, DIV, DO, DUPLICATE, HOUR_MILLISECOND, HOUR_SECOND, HOUR_MINUTE, LOCALTIME, LOCALTIMESTAMP, MINUTE_MILLISECOND, MINUTE_SECOND, MOD, ROLLUP, SECOND_MILLISECOND, TRUNCATE, XOR, YEAR_MONTH
예약어를 식별자로 사용하기 위한 처리 부호	큰따옴표(" ), 대괄호([ ]), 그레이브 부호(`) CREATE TABLE "DATABASE"; CREATE TABLE [DATABASE]; CREATE TABLE 'DATABASE';
식별자에 포함할 수 없는 특수 문자	%, !,  , ^, ~

### ● 응용 프로그램의 수정 필요 여부 검토

일부 함수 및 SQL 질의 수행 방식이 변경되어 질의 결과가 이전 버전과 다를 수 있으므로 주의하여야 한다. 이전 버전과 다른 결과를 반환하는 질의는 다음과 같다.

이전과 다른 결과를 반환하는 질의 유형	이전 버전	CUBRID 2008 R3.0	상세 정보
AUTO_INCREMENT 컬럼에 NULL 입력되는 경우	NULL 저장	<p>자동 증가 값 저장</p> <pre>CREATE TABLE auto_tbl (id INT AUTO_INCREMENT, name VARCHAR); INSERT INTO auto_tbl VALUES (NULL, 'AAA'), (NULL, 'BBB'), (NULL, 'CCC'); INSERT INTO auto_tbl (name) VALUES ('DDD'), ('EEE'); SELECT * FROM auto_tbl; ;xr</pre> <p>=== &lt;Result of SELECT Command in Line 4&gt; ===</p> <pre>      id  name =====       1  'AAA'       2  'BBB'       3  'CCC'       4  'DDD'       5  'EEE'  5 rows selected.</pre>	<a href="#">바로가기</a>
외래 키에서 ON UPDATE/ON DELETE ACTION으로 NO ACTION이 정의된 경우	아무 동작하지 않음	RESTRICT와 동일하게 동작	<a href="#">바로가기</a>
POWER 함수의 반환 값 타입	인자 타입에 따라 다름	DOUBLE 타입	<a href="#">바로가기</a>

TO_CHAR 함수에서 'HH' 또는 'HH12' 지정자로 12시간 포맷을 사용하는 경우	표현 범위: 0~11	표현 범위: 1~12	<a href="#">바로가기</a>																																				
두 자리 수 연도 값의 변환 결과	01~99 → 0001 ~ 0099	00~69 → 2000~2069 70~99 → 1970~1999	<a href="#">바로가기</a>																																				
날짜/시간 데이터 타입의 비교/산술 연산	동일한 타입에 대해서만 비교 가능 동일한 타입에 대해서만 뺄셈 가능	<div>DATE, TIMESTAMP, DATETIME 타입 간 비교 가능</div> <div>DATE, TIMESTAMP, DATETIME 타입 간 뺄셈 가능</div> <div><pre>SELECT DATETIME '09/01/2009 03:30:30.001 pm' - TIMESTAMP '08/31/2009 03:30:30 pm';</pre><div>=====</div><div>86400001 (밀리초 단위)</div><div><pre>SELECT DATETIME '09/01/2009 03:30:30.001 pm' + 86400000;</pre><div>=====</div><div>2009-09-02 15:30:30.000</div><div><pre>SELECT TIMESTAMP '09/01/2009 03:30 pm' + 86400;</pre><div>=====</div><div>2009-09-02 15:30:00</div></div><div><div>피연산자의 타입별 허용 연산과 결과 데이터 타입</div><table><tr><th></th><th>TIME (초 단위)</th><th>DATE (일 단위)</th><th>TIMESTAMP (초 단위)</th><th>DATETIME (밀리초 단위)</th><th>INT</th></tr><tr><th>TIME</th><td>뺄셈 INT</td><td>X</td><td>X</td><td>X</td><td>덧셈, 뺄셈 TIME</td></tr><tr><th>DATE</th><td>X</td><td>뺄셈 INT</td><td>뺄셈 INT</td><td>뺄셈 INT</td><td>덧셈, 뺄셈 DATE</td></tr><tr><th>TIMESTAM P</th><td>X</td><td>뺄셈 INT</td><td>뺄셈 INT</td><td>뺄셈 INT</td><td>덧셈, 뺄셈 TIMESTAMP</td></tr><tr><th>DATETIM E</th><td>X</td><td>뺄셈 INT</td><td>뺄셈 INT</td><td>뺄셈 INT</td><td>덧셈, 뺄셈 DATETIME</td></tr><tr><th>INT</th><td>덧셈, 뺄셈 TIME</td><td>덧셈, 뺄셈 DATE</td><td>덧셈, 뺄셈 TIMESTAMP</td><td>덧셈, 뺄셈 DATETIME</td><td>모든 연산</td></tr></table></div></div></div>		TIME (초 단위)	DATE (일 단위)	TIMESTAMP (초 단위)	DATETIME (밀리초 단위)	INT	TIME	뺄셈 INT	X	X	X	덧셈, 뺄셈 TIME	DATE	X	뺄셈 INT	뺄셈 INT	뺄셈 INT	덧셈, 뺄셈 DATE	TIMESTAM P	X	뺄셈 INT	뺄셈 INT	뺄셈 INT	덧셈, 뺄셈 TIMESTAMP	DATETIM E	X	뺄셈 INT	뺄셈 INT	뺄셈 INT	덧셈, 뺄셈 DATETIME	INT	덧셈, 뺄셈 TIME	덧셈, 뺄셈 DATE	덧셈, 뺄셈 TIMESTAMP	덧셈, 뺄셈 DATETIME	모든 연산	<a href="#">바로가기</a>
	TIME (초 단위)	DATE (일 단위)	TIMESTAMP (초 단위)	DATETIME (밀리초 단위)	INT																																		
TIME	뺄셈 INT	X	X	X	덧셈, 뺄셈 TIME																																		
DATE	X	뺄셈 INT	뺄셈 INT	뺄셈 INT	덧셈, 뺄셈 DATE																																		
TIMESTAM P	X	뺄셈 INT	뺄셈 INT	뺄셈 INT	덧셈, 뺄셈 TIMESTAMP																																		
DATETIM E	X	뺄셈 INT	뺄셈 INT	뺄셈 INT	덧셈, 뺄셈 DATETIME																																		
INT	덧셈, 뺄셈 TIME	덧셈, 뺄셈 DATE	덧셈, 뺄셈 TIMESTAMP	덧셈, 뺄셈 DATETIME	모든 연산																																		
JDBC 에서 URL String 입력 정보	속성(PROPERTY) 정보 앞에 물음표 생략 가능	속성(PROPERTY) 정보 앞에 반드시 물음표 명시	<a href="#">바로가기</a>																																				

#### ● 파라미터 변경 사항 확인

아래의 파라미터는 변경 사항이 있으므로, 환경 설정 파일을 적용하는 경우 주의한다.

파라미터 이름	변경 사항	상세 정보
INSERT_EXECUTION_MODE	<p>파라미터 값 설정 범위 확장</p> <p>기본 : INSERT SELECT(1), INSERT VAULES(2), INSERT DEFAULT(4)</p> <p>확장 : INSERT REPLACE(8), INSERT ON DUP KEY UPDATE(16)</p>	<a href="#">바로가기</a>

	<pre> ➤ CREATE TABLE code2(s_name char(1) NOT NULL UNIQUE,   f_name varchar(40));   REPLACE INTO code2 VALUES ('S', 'Silver');  ➤ CREATE TABLE code2(s_name char(1) NOT NULL UNIQUE,   f_name varchar(40));   INSERT INTO code2 VALUES ('S', 'Silver') ON DUPLICATE   KEY UPDATE f_name='Silver'; </pre>	
ANSI_QUOTES	추가됨 (디폴트 값 yes)	<a href="#">바로가기</a>
PIPES_AS_CONCAT	추가됨 (디폴트 값 yes)	
ONLY_FULL_GROUP_BY	추가됨 (디폴트 값 no)	

### ● 기존 환경 설정 파일 보관

이전 버전의 \$CUBRID/conf 디렉터리의 환경 설정 파일(**cubrid.conf**, **cubrid\_broker.conf**, **cm.conf**)과 \$CUBRID\_DATABASES 디렉터리의 데이터베이스 위치 정보 파일(**databases.txt**)을 보관한다.

### ● 데이터베이스 마이그레이션

CUBRID 2008 R3.0 버전은 하위 버전의 데이터베이스와 호환되지 않으므로, 기존 데이터베이스를 CUBRID 2008 R3.0 버전으로 마이그레이션 하여야 한다. ([데이터베이스 마이그레이션 절차 참고](#))

### ● 복제 또는 HA 환경 재구성

이전 버전의 복제 기능을 사용하는 시스템에서는 보다 안정적인 운영을 위해 DB 마이그레이션 및 HA 환경으로 재구성할 것을 권장한다. 또한, CUBRID 2008 R2.2 이전 버전에서 제공된 Linux Heartbeat 기반의 HA 기능을 사용하는 시스템도 보다 안정적인 운영을 위해 DB 마이그레이션 및 CUBRID Heartbeat 기반의 HA 환경으로 재구성할 것을 권장한다. ([HA 환경에서 마이그레이션 절차 참고](#))

## 데이터베이스 마이그레이션 절차

<CUBRID 설치 디렉토리>/bin/migrate\_r30 유틸리티를 이용하여 마이그레이션을 수행할 수 있으며, 아래의 수행 가이드를 참고한다. 이때, 다운로드 페이지에서 별도 배포되는 **check\_reserved.sql** 예약어 검출 스크립트를 이용하여 기존 버전의 데이터베이스에 사용된 식별자 중 CUBRID 2008 R3.0에서 추가된 예약어가 존재하는지를 검출할 수 있다.

CUBRID 업그레이드 이후 \$CUBRID/share/scripts 경로에 존재 파일이 존재한다.

다른 방법으로는 **cubrid unloaddb/loaddb** 유틸리티를 사용하여 마이그레이션을 수행할 수 있으며, 이는 [관리자 안내서> 데이터베이스 마이그레이션](#)을 참고한다.

단계	Linux 환경	Windows 환경
C1 단계: CUBRID Service 종료	% cubrid service stop	CUBRID 서비스 트레이>[Exit] 선택
C2 단계: 예약어 검출 스크립트 실행	<p>예약어 검출 스크립트가 위치하는 디렉토리에서 아래 명령을 실행한다. 검출 결과를 확인하여 마이그레이션 진행 또는 식별자 수정 작업을 진행한다. <a href="#">SQL 설명서&gt; 식별자</a>를 참고한다.</p> <p>아래와 같이 예약어가 검출될 경우 응용질의 수행 시 정의한 스키마에 대하여 예약어 식별자를 부여하여 질의를 변경하여야 한다.</p> <pre> % csql -S -u dba -i check_reserved.sql &lt;r22_db_name&gt;  R30 new reversed keyword ===== </pre>	

	<pre>{'database', 'day_millisecond', 'day_second', 'day_minute', 'day_hour', 'distinctrow', 'div', 'do', 'duplicate', 'hour_millisecond', 'hour_second', 'hour minute', 'localtime', 'localtimestamp', 'minute_millisecond', 'minute second', 'mod', 'rollup', 'second_millisecond', 'truncate', 'xor', 'year month'}</pre> <p>=== &lt;Result of SELECT Command in Line 50&gt; ===</p> <pre>object          name          from table from method or storedproc or trigger ===== = 'column'        'xor'          'test01'      '' 'column'        'do'           'test01'      '' 'column'        'distinctrow'  'test01'      ''</pre>	
C3 단계: 이전 버전 DB 백업	<p>2008 R3.0 운영 중 이전 버전으로 복구하는 상황에 대비하기 위해 백업을 수행하고, 백업 파일을 별도 디렉터리(R22_backup)에 보관한다. (C2a)</p> <pre>% mkdir R22_backup % cubrid backupdb -S -D R22_backup &lt;r22_db_name&gt;</pre> <p>이전 버전의 databases.txt 및 conf 디렉토리 내 설정 파일을 별도 디렉터리에 보관한다. (C2b)</p>	
C4 단계: 2008 R3.0 버전 설치	<p>본 문서의 <a href="#">CUBRID 2008 R3.0의 설치 방법</a>을 참고한다.</p>	
C5 단계: 마이그레이 션 도구 실행		<p>CUBRID 서비스 트레이&gt; [CUBRID Server]&gt;[Stop] 선택하여 서버 종료</p> <p>(C2b)에서 보관한 databases.txt를 2008 R3.0의 설치 디렉터리에 복사한다. (C5a)</p> <p>아래와 같이 migrate_r30 유틸리티를 실행한다. (C5b)</p> <pre>% migrate r30 &lt;db name&gt;</pre>
C6 단계: 2008 R3.0 버전 DB 백업	<p>데이터베이스 버전이 R1.x인 경우에만 수행한다. 이전 버전이 R2.x 버전인 경우, 이 단계를 생략할 수 있다.</p> <pre>% cubrid backupdb -S &lt;db_name&gt;</pre>	
C7 단계: CUBRID 환경 설정 및 CUBRID Service 구동	<p>환경 설정 파일을 수정한다. 이때, (C2b)에서 보관한 이전 버전의 환경 설정 파일을 사용할 수 있다.</p> <pre>% cubrid service start % cubrid server start &lt;db_name&gt;</pre>	<p>CUBRID 서비스 트레이&gt; [CUBRID Server]-&gt; [Start]</p>

## HA 환경에서 데이터베이스 마이그레이션 절차

CUBRID 2008 R2.0, R2.1 버전의 HA 기능을 사용하는 경우, 해당 버전에서 사용되었던 Linux Heartbeat 패키지가 더 이상 사용되지 않으므로 아래 가이드를 참고하여 서버 버전 업그레이드, DB 마이그레이션을 수행한 후 HA 환경을 새롭게 구축하여야 한다. 한편, CUBRID 2008 R2.2 이상 버전의 HA 기능을 사용하는 경우에는 서버 업그레이드, DB 마이그레이션만 수행하면 된다.

아래는 브로커, 마스터 DB, 슬레이브 DB를 각각 별도 서버에 구축한 환경에서 현재 서비스를 중지하고 업그레이드를 수행하기 위한 가이드이다. 서비스 무정지 업그레이드 시나리오는 별도 가이드 문서를 참고한다.

단계	설명
H1 단계: HA 관련 서비스 종료 및 기존 Linux heartbeat 제거	<pre>% cubrid broker stop % service heartbeat stop % chkconfig --del heartbeat</pre>

	<pre>% pkill -u user1 -f "cub_master"</pre>
H2~H6 단계: 마스터 서버에서 C2~C6 단계를 수행	마스터 서버에서 CUBRID 업그레이드 및 데이터베이스 마이그레이션을 수행하고, 2008 R3.0 데이터베이스를 백업한다.
H7 단계: 슬레이브 서버에 CUBRID 2008 R3.0 버전 설치	설치 방법은 본 문서의 <a href="#">CUBRID 2008 R3.0의 설치 방법</a> 을 참고한다.
H8 단계: 마스터 백업본을 슬레이브 서버에서 복구	H6에서 생성된 마스터 서버의 2008 R3.0 데이터베이스 백업본(testdb_bk*)을 슬레이브 서버에서 복구한다. <pre>% scp user1@m_server:~/DB/testdb/testdb bk0v000 . % scp user1@m_server:~/DB/testdb/log/testdb bkvinf ./log/. % cubrid restoredb testdb</pre>
H9 단계: HA 환경 재구성 후 HA 모드 구동	HA 구동 스크립트(cubrid-ha) 및 환경 설정 파일(cubrid.conf)을 설정한다. <a href="#">관리자 안내서&gt; CUBRID HA 환경 설정</a> 을 참고한다. 마스터 서버 및 슬레이브 서버에서 HA 모드로 DB 를 구동한다. <a href="#">관리자 안내서&gt; CUBRID HA 모드 구동</a> 을 참고한다. <pre>[root@m_server~]# service cubrid-ha start [root@s_server~]# service cubrid-ha start</pre>
H10 단계: 브로커 서버에 CUBRID 2008 R3.0 버전 설치 및 브로커 구동	설치 방법은 본 문서의 <a href="#">CUBRID 2008 R3.0의 설치 방법</a> 을 참고한다. 브로커 설정 후 브로커를 시작한다. <a href="#">관리자 안내서&gt; CUBRID HA 환경 설정</a> 을 참고한다. <pre>% cat cubrid broker.conf ... ACCESS_MODE=RW  % cubrid broker start</pre>

## 2. CUBRID 2008 R3.0 버전에서 변경된 사항

### 새로 추가된 기능 - SQL 문법 확장 관련

#### CUBRIDSUS-3590 CREATE TABLE 문 확장 및 테이블 복제 기능 지원

- 기존 테이블과 동일한 스키마를 가지는 테이블을 생성하는 기능 지원

기존 테이블의 스키마만 복제하여 새로운 테이블을 생성할 수 있도록 **CREATE TABLE** 문법을 확장하였다. **CREATE TABLE ... LIKE** 문을 사용하여 보다 편리하게 새로운 테이블의 스키마를 정의할 수 있다.

```
// a_tbl 테이블의 스키마만 복제하여 새로운 테이블 생성
CREATE TABLE new_code LIKE code;
csql> ;sc new code
<Class Name>

    new code

<Attributes>

    s_name          CHARACTER(1)
    f_name          CHARACTER VARYING(6)
```

⇒ 분할 테이블, AUTO\_INCREMENT 컬럼이 포함된 테이블, 상속 또는 메소드를 사용하는 테이블

- 기존 테이블에 대한 **SELECT** 부질의 결과 레코드를 포함하는 테이블을 생성하는 기능 지원

다수 개의 레코드가 삽입된 테이블이 존재하는 경우, **SELECT** 문의 결과 레코드를 포함하는 새로운 테이블을 생성할 수 있도록 **CREATE TABLE** 문법을 확장하였다. **CREATE TABLE ... AS SELECT** 문을 사용하여 보다 편리하게 새로운 테이블의 스키마 정의 및 레코드 삽입을 수행할 수 있다.

```
// a_tbl 테이블에 대한 SELECT 결과를 복제하여 새로운 테이블 생성
CREATE TABLE a_tbl(
id INT NOT NULL DEFAULT 0 PRIMARY KEY,
phone VARCHAR(10));

INSERT INTO a_tbl VALUES(1,'111-1111'), (2,'222-2222'), (3, '333-3333');

--creating a table without column definition
CREATE TABLE new_tbl1 AS SELECT * FROM a_tbl;
SELECT * FROM new_tbl1;
;x

=== <Result of SELECT Command in Line 1> ===

    id  phone
=====
    1  '111-1111'
    2  '222-2222'
    3  '333-3333'

--all of column values are replicated from a_tbl
CREATE TABLE new_tbl2
(id INT NOT NULL AUTO INCREMENT PRIMARY KEY, phone VARCHAR) AS SELECT * FROM a_tbl;
SELECT * FROM new_tbl2;
;x

=== <Result of SELECT Command in Line 1> ===

    id  phone
=====
    1  '111-1111'
    2  '222-2222'
    3  '333-3333'

--some of column values are replicated from a_tbl and the rest is NULL
```

```
CREATE TABLE new_tbl3
(id INT, name VARCHAR) AS SELECT id, phone FROM a_tbl;
SELECT * FROM new_tbl3
;X
```

=== <Result of SELECT Command in Line 1> ===

name	id	phone
NULL	1	'111-1111'
NULL	2	'222-2222'
NULL	3	'333-3333'

--column alias in the select statement should be used in the column definition

```
CREATE TABLE new_tbl4
(id1 int, id2 int) AS SELECT t1.id id1, t2.id id2 FROM new_tbl1 t1, new_tbl2 t2;
SELECT * FROM new_tbl4;
;X
```

=== <Result of SELECT Command in Line 1> ===

id1	id2
1	1
1	2
1	3
2	1
2	2
2	3
3	1
3	2
3	3

--REPLACE is used on the UNIQUE column

```
CREATE TABLE new_tbl5(id1 int UNIQUE) REPLACE AS SELECT * FROM new_tbl4;
SELECT * FROM new_tbl5;
;X
```

=== <Result of SELECT Command in Line 1> ===

id1	id2
1	3
2	3
3	3

→ id2 컬럼 값의 결정은 가장 마지막에 들어온 값

## ● 테이블 생성 시 인덱스 생성 기능 지원

테이블 생성 시 컬럼 정의문 내에서 **INDEX, KEY, UNIQUE INDEX** 를 정의하여 인덱스를 생성할 수 있도록 **CREATE TABLE** 문법을 확장하였다 그리고, 인덱스 정의 시 대상 컬럼 이름 뒤에 **ASC** 또는 **DESC** 옵션을 명시할 수 있으며, **GROUP BY** 절 내의 컬럼에 대해서도 컬럼 옵션을 명시하여 간편하게 정렬을 수행할 수 있다.

```
// 테이블 생성 시 인덱스 정의
CREATE TABLE new_tbl(
id INT,
name VARCHAR,
CONSTRAINT UNIQUE INDEX(id DESC, name ASC)
);
```

- 관련 문서: [테이블 정의> CREATE TABLE, 컬럼 옵션](#)

## CUBRIDSUS-3590 prefix\_length에 해당하는 앞 부분 스트링에 대해 인덱스 생성 기능 지원 및 시스템 카탈로그 테이블 변경

문자열 타입 컬럼에 인덱스를 정의하는 경우, prefix\_length 를 명시하여 스트링의 앞 부분(prefix) 일부에 대해서만 인덱스를 생성할 수 있다. 이 기능과 관련하여 시스템 카탈로그 가상 테이블 **DB\_INDEX\_KEY** 에 prefix\_length 값이 저장되는 key\_prefix\_length 컬럼이 추가되었다.

```
CREATE TABLE new_tbl ( name VARCHAR(20));
```



```

INSERT INTO new_tbl VALUES ('aba1234'), ('abb1234'), ('abc1234'), ('abd1234'), ('abe1234'),
('abf1234'), ('abg1234');
// 스트링 타입 컬럼에 대해 3 바이트 길이 prefix에 대해 인덱스 생성
CREATE INDEX ON new_tbl(name(3));

name
=====
'aba1234'
'abb1234'
'abc1234'
'abd1234'
'abe1234'
'abf1234'
'abg1234'

// 시스템 카탈로그에서 정의된 prefix_length 값 조회
SELECT class_name, key_attr_name, key_prefix_length FROM db_index_key
WHERE class_name = 'new_tbl';

```

- 관련 문서: [인덱스 정의](#)> [CREATE INDEX](#), [시스템 카탈로그 가상 클래스](#)> [DB\\_INDEX\\_KEY](#)

## CUBRIDSUS-3590 INSERT 문법 확장 및 REPLACE 문 지원

- 여러 개의 레코드를 한번에 삽입할 수 있도록 INSERT 문법 확장

INSERT 문에서 INTO 를 생략할 수 있으며, VALUES 대신 VALUE 를 사용할 수도 있다. 또한, 삽입하고자 하는 값을 콤마로 구분하여 여러 행을 한번에 삽입할 수 있도록 구문을 확장하였다.

```

CREATE TABLE insert_tbl (A INT DEFAULT 1, B INT UNIQUE);
INSERT INTO insert_tbl VALUES (1,1), (2,2), (NULL, 3);

```

- 디폴트 값을 간편하게 삽입할 수 있도록 구문 확장

DEFAULT 키워드를 VALUES 앞 또는 뒤에 명시하거나, 컬럼 값에 직접 명시할 수 있도록 구문을 확장하였다.

```

// 정의된 디폴트 값 삽입
INSERT INTO insert_tbl VALUES DEFAULT;
INSERT INTO insert_tbl DEFAULT VALUES;

=====
a          b
=====
1          1
2          2
NULL       3
1          NULL
1          NULL

// 컬럼 값에 DEFAULT 키워드 직접 명시
INSERT INTO insert_tbl VALUES (DEFAULT, 4), (DEFAULT, 5);

=====
a          b
=====
1          1
2          2
NULL       3
1          NULL
1          NULL
1          4
1          5

```

- 특정 컬럼 값을 간편하게 삽입할 수 있도록 SET 절 지원

INSERT 문에서 VALUES 절 대신 SET 절을 사용하여 삽입하고자 하는 특정 컬럼 이름 및 컬럼 값을 지정할 수 있다.

```

// 일부 컬럼 값만 삽입
INSERT INTO insert_tbl SET a=10;

=====
a          b
=====
1          1
2          2
NULL       3

```

1	NULL
1	NULL
1	4
1	5
10	NULL

- **UNIQUE 컬럼에 중복된 값을 삽입할 수 있도록 ON DUPLICATE KEY UPDATE 절 지원**

**INSERT** 문에서 **ON DUPLICATE KEY UPDATE** 절을 사용하여 **UNIQUE** 또는 **PRIMARY KEY** 제약 조건을 위반하는 레코드가 삽입되더라도 컬럼 값을 새로운 값으로 갱신할 수 있다.

```
// insert 시 중복 제약조건에 위배될 경우 update 수행
// update 가 되는 Row 값이 동일할 경우 오류 및 데이터변경 없음
// insert_tbl 테이블의 unique 한 컬럼 b에 중복된 5의 값을 insert
INSERT INTO insert_tbl VALUES (DEFAULT, 5) ON DUPLICATE KEY UPDATE A=2;

insert tbl

<Attributes>

a          INTEGER DEFAULT 1
b          INTEGER

<Constraints>

UNIQUE u_insert_tbl_b ON insert_tbl (b)

=====
a          b
=====
1          1
2          2
NULL      3
1          NULL
1          NULL
1          4
2          5
10         NULL
```

- **레코드 삭제 후 삽입을 수행하는 REPLACE 문 추가 지원**

**INSERT** 문과 유사하지만, **ON DUPLICATE KEY UPDATE** 절을 명시하지 않고도 **UNIQUE** 또는 **PRIMARY KEY** 제약 조건을 위반하는 레코드가 삽입되더라도 해당 컬럼 값을 새로운 값으로 갱신할 수 있는 **REPLACE** 문을 지원한다.

```
// 중복된 값 삽입 시 기존 레코드 삭제 후 새로운 레코드가 삽입
REPLACE INTO insert_tbl VALUES (3, 5);

=====
a          b
=====
1          1
2          2
NULL      3
1          NULL
1          NULL
1          4
10         NULL
3          5
```

- 관련 문서: [데이터 조회 및 조작 > INSERT, ON DUPLICATE KEY UPDATE 절, REPLACE](#)

## CUBRIDSUS-3590 SELECT 문, DELETE 문, UPDATE 문에서 대상 레코드의 개수를 제한하는 LIMIT 절 지원

**LIMIT** 절을 사용하여 **SELECT** 문에서 실행 결과 셋의 레코드 개수를 제한할 수 있다. 마찬가지로 **DELETE** 문 및 **UPDATE** 문에서도 연산을 수행할 레코드의 개수를 제한할 수 있다.

```
//결과 셋의 2 번째 레코드부터 3 개만 출력
SELECT * FROM a_tbl ORDER BY a LIMIT 2,3;

//조건을 만족하는 레코드 중 처음 3 개만 업데이트
```

```
UPDATE a_tbl SET a=10, b=10 WHERE a IS NULL LIMIT 3;

//조건을 만족하는 레코드 중 처음 1 개만 삭제
DELETE FROM a_tbl WHERE a IS NULL LIMIT 1;
```

- 관련 문서: [데이터 조회 및 조작> SELECT> LIMIT 절](#)

## CUBRIDSUS-3590 SELECT 문에서 FROM 절 생략 가능

**SELECT** 문에서 참조하는 테이블이 없는 경우 이를 생략할 수 있도록 문법을 확장하였다.

```
SELECT 1+1;
      1+1
=====
      2

SELECT FALSE;
      false
=====
      0

SELECT TRUE;
      true
=====
      1
```

- 관련 문서: [데이터 조회 및 조작> SELECT> FROM 절](#)

## CUBRIDSUS-3590 SELECT 문에서 GROUP BY ... HAVING 절의 문법 확장

### 지원

**GROUP BY** 절에서 컬럼 이름 대신 컬럼 별칭을 명시할 수 있으며, 그룹 별 중간 집계 및 총 집계를 출력할 수 있도록 **WITH ROLLUP** 구문을 지원한다. 또한, **ORDER BY NULL** 구문을 명시하여 **GROUP BY** 절에 의한 정렬이 수행되지 않도록 할 수 있다.

한편, **ONLY\_FULL\_GROUP\_BY** 파라미터 설정 값이 NO인 경우에는, **GROUP BY** 절에 명시하지 않은 컬럼을 **SELECT** 리스트에서 사용할 수 있도록 확장된 문법을 제공한다.

```
CREATE TABLE sales_tbl
(dept_no int,
 name VARCHAR(20) PRIMARY KEY,
 sales_month int,
 sales_amount int DEFAULT 100);

INSERT INTO sales_tbl VALUES
(201, 'George' , 1, 450),
(201, 'Laura' , 2, 500),
(301, 'Max' , 4, 300),
(501, 'Stephan', 4, DEFAULT),
(501, 'Chang' , 5, 150),
(501, 'Sue' , 6, 150),
(NULL, 'Yoka' , 4, NULL);

// only_full_group_by 파라미터 값이 NO(디폴트)일 경우, sales_amount에 대하여 SELECT에 포함
// GROUP 별 avg(sales_amount) 값을 구한다.
// ORDER BY NULL에 의해서 dept_no에 대하여 group by 시 정렬하지 않음
SELECT dept_no, avg(sales_amount) FROM sales_tbl
GROUP BY dept_no ORDER BY NULL;

      dept_no  avg(sales amount)
=====
      NULL      NULL
      201      475
      301      300
      501      133

// WHERE 조건을 수행한 후 GROUP BY 수행
// dept_no 값이 501인 ROW 중 null을 포함하여 위와 같이 평균이 133이 없으나, WHERE 절을 선행 수행하여 평균 150이 됨
SELECT dept_no, avg(sales amount) FROM sales_tbl
WHERE sales_amount > 100 GROUP BY dept_no;
```

```

dept_no avg(sales_amount)
=====
201      475
301      300
501      150

// group by 후에 HAVING 절을 수행
SELECT dept_no, avg(sales_amount) FROM sales_tbl
WHERE sales_amount > 100 GROUP BY dept_no HAVING avg(sales_amount) > 200;

dept no avg(sales amount)
=====
201      475
301      300

--select 의 alias 이용하여 GROUP BY, HAVING, ORDER BY 수행
--단, WHERE 절의 alias 사용할 수 없음
SELECT dept_no AS a1, avg(sales_amount) AS a2 FROM sales_tbl
WHERE sales amount > 200 GROUP BY a1 HAVING a2 > 200 ORDER BY a2;

a1      a2
=====
301      300
201      475

SELECT dept no AS a1, avg(sales amount) AS a2 FROM sales tbl
WHERE a2 > 200 GROUP BY a1 HAVING a2 > 200 ORDER BY a2;

ERROR: a2 is not defined.

--그룹별 평균 sales_amount 를 보기 위해 WITH ROLLUP(중간집계) 추가하여 수행
SELECT dept_no AS a1, name AS a2, AVG(sales_amount) AS a3 FROM sales_tbl
WHERE sales amount > 100 GROUP BY a1,a2 WITH ROLLUP;

a1 a2      a3
=====
201 'George' 450
201 'Laura' 500
201 NULL    475
301 'Max'   300
301 NULL    300
501 'Chang' 150
501 'Sue'   150
501 NULL    150
NULL NULL    310 → 중간 집계를 제외한 전체평균

SELECT dept_no AS a1, name AS a2, MAX(sales_amount) AS a3 FROM sales_tbl
WHERE sales_amount > 100 GROUP BY a1,a2 WITH ROLLUP;

a1 a2      a3
=====
201 'George' 450
201 'Laura' 500
201 NULL    500
301 'Max'   300
301 NULL    300
501 'Chang' 150
501 'Sue'   150
501 NULL    150
NULL NULL    500

```

- 관련 문서: [데이터 조회 및 조작 > SELECT > GROUP BY HAVING 절](#)

## CUBRID SUS-3590 ALTER TABLE 문법 확장 및 추가할 컬럼 위치 지정

### 기능 지원

- 인덱스 및 제약 조건 추가 및 삭제 기능 지원

ALTER TABLE 문을 사용하여 보다 간편하게 인덱스 및 제약 조건 정의를 변경할 수 있도록 ADD INDEX 절, DROP INDEX 절, DROP PRIMARY KEY 절, DROP FOREIGN KEY 절을 추가 지원한다.

```

<Class Name>
    sales_tbl

<Attributes>
    dept no          INTEGER
    name             CHARACTER VARYING(20) NOT NULL
    sales month      INTEGER
    sales_amount     INTEGER DEFAULT 100

<Constraints>
    PRIMARY KEY pk sales tbl name ON sales tbl (name)

// ALTER TABLE 문 내에서 인덱스 추가
ALTER TABLE sales tbl ADD INDEX (dept no ASC), ADD INDEX (name DESC);
<Constraints>
    INDEX i_sales_tbl_dept_no ON sales_tbl (dept_no)
    INDEX i_sales_tbl_name_d ON sales_tbl (name DESC)
    PRIMARY KEY pk sales tbl name ON sales tbl (name)

// ALTER TABLE 문 내에서 인덱스 삭제
ALTER TABLE sales_tbl DROP INDEX i_sales_tbl_dept_no;
<Constraints>
    INDEX i_sales_tbl_name_d ON sales_tbl (name DESC)
    PRIMARY KEY pk_sales_tbl_name ON sales_tbl (name)

// ALTER TABLE 문 내에서 기본 키 삭제
ALTER TABLE sales tbl DROP PRIMARY KEY;
<Constraints>
    INDEX i_sales_tbl_name_d ON sales_tbl (name DESC)

ALTER TABLE sales tbl ADD PRIMARY KEY (name);
<Constraints>
    PRIMARY KEY pk_sales_tbl_name ON sales_tbl (name);

```

- 추가할 컬럼 위치 지정 기능 추가

**ALTER TABLE ... ADD COLUMN** 절을 사용하여 새로운 컬럼을 추가하는 경우, **FIRST** 또는 **AFTER** 키워드를 사용하여 해당 컬럼의 위치를 지정할 수 있는 기능을 지원한다.

```

ALTER TABLE sales_tbl ADD COLUMN cost INT DEFAULT 0 AFTER name;
<Attributes>
    dept_no          INTEGER
    name             CHARACTER VARYING(20)
    cost             INTEGER DEFAULT 0
    sales month      INTEGER
    sales_amount     INTEGER DEFAULT 100

ALTER TABLE sales_tbl ADD COLUMN price INT DEFAULT 0 FIRST;
<Attributes>
    price            INTEGER DEFAULT 0
    dept no          INTEGER
    name             CHARACTER VARYING(20)
    cost             INTEGER DEFAULT 0
    sales month      INTEGER
    sales_amount     INTEGER DEFAULT 100

```

- 관련 문서: [테이블 정의> ALTER TABLE](#)

## CUBRIDSUS-3590 TRUNCATE 문을 사용한 모든 레코드 삭제 기능 지원

**TRUNCATE** 문을 사용하여 테이블 내 모든 레코드를 삭제할 수 있으며, 이는 대상 테이블에 정의된 모든 인덱스와 제약 조건을 먼저 삭제한 후 레코드를 삭제하므로, 기존 **DELETE** 문을 사용하는 경우보다 성능 상 유리하다.

```
TRUNCATE sales_tbl;
```

- 관련 문서: [데이터 조회 및 조작> TRUNCATE](#)

## CUBRIDSUS-3590 외래 키 정의 시 ON UPDATE 및 ON DELETE 연산에 대한 트리거 동작 추가

기본 키 값이 업데이트 또는 삭제되는 경우, 이를 참조하는 외래 키 값을 **NULL**로 갱신하는 **SET NULL** 동작 옵션을 추가 지원한다. 한편, **NO ACTION** 동작 옵션의 경우 이전 버전에서는 기본 키 값의 변경을 허용하고 외래 키에 대해서는 아무런 동작을 수행하지 않았으나, 이를 **RESTRICT** 동작 옵션과 동일하게 동작하도록 수정하였다. 즉, **NO ACTION** 또는 **RESTRICT** 동작 옵션이 정의된 경우, 외래 키가 참조하는 기본 키 값을 변경할 수 없다.

```
<referential_triggered_action_on_update> ::=
ON UPDATE { RESTRICT | NO ACTION | SET NULL }

<referential_triggered_action_on_delete> ::=
ON DELETE { CASCADE | RESTRICT | NO ACTION | SET NULL }
```

**CASCADE** : 기본키가 삭제되면 외래키도 삭제한다. ON DELETE 연산에 대해서만 지원된다.

**RESTRICT** : 기본키 값이 삭제되거나 업데이트되지 않도록 제한한다. 삭제 또는 업데이트를 시도하는 트랜잭션은 롤백 된다.

**SET NULL** : 기본키가 삭제되거나 업데이트되면, 이를 참조하는 외래키 컬럼 값을 NULL로 업데이트한다.

**NO ACTION** : **RESTRICT** 옵션과 동일하게 동작한다.

- 관련 문서: [테이블 정의> CREATE TABLE> FOREIGN KEY 제약 조건](#)

## CUBRIDSUS-3590 SQL 레벨에서 PREPARED STATEMENT 실행 기능 지원

미리 SQL 문을 선언하는 **PREPARE** 문, 이를 실행하는 **EXECUTE** 문, 선언된 SQL 문을 삭제하는 **DROP PREPARE** 문을 지원하며, 이를 사용하여 prepared statement 기능을 사용할 수 있다. 단, CUBRID 매니저의 질의 편집기에서는 이 기능이 지원되지 않으며, CSQL 인터프리터에서는 **PREPARE** 문과 **EXECUTE** 문을 함께 작성한 후 실행 명령어(**;<xr 또는 ;run**)를 입력하여야 한다.

```
csql> PREPARE stmt1 FROM 'SELECT POWER(?,2)*PI()';
csql> EXECUTE stmt1 USING 2;
csql> DROP PREPARE stmt1;
csql> PREPARE stmt1 FROM 'SELECT POWER(?,2)*PI()';
csql> EXECUTE stmt1 USING 2;
csql> ;x
```

Current transaction has been committed.

=== <Result of SELECT Command in Line 2> ===

```
power( ?:0 , 2)* pi()
=====
1.256637061435917e+001
```

1 rows selected.

Current transaction has been committed.

Current transaction has been committed.

Current transaction has been committed.

=== <Result of SELECT Command in Line 5> ===

```
power( ?:0 , 2)* pi()
=====
1.256637061435917e+001
```

- 관련 문서: [데이터 조회 및 조작> PREPARED STATEMENT](#)

## CUBRIDSUS-3590 SQL 확장 문법 적용을 위한 파라미터 추가 지원

SQL의 확장 문법 또는 표준 문법 적용을 위한 파라미터를 추가로 지원한다. 응용 프로그램에서 작성하고자 하는 SQL 유형 및 코딩 스타일에 따라 해당하는 파라미터를 설정할 수 있다.

추가된 파라미터	파라미터 값(디폴트 설정)	파라미터 값
ONLY_FULL_GROUP_BY	NO  확장 문법 적용.  GROUP BY 절에 명시되지 않은 컬럼을 SELECT 컬럼 리스트에 명시 가능	YES  SQL 표준 문법을 적용.  GROUP BY 절에 명시한 컬럼만 SELECT 컬럼 리스트에 명시 가능
ANSI_QUOTES	YES  작은 따옴표(')를 문자열 처리 부호로 사용	NO  큰 따옴표(")도 문자열 처리 부호로 사용
PIPES_AS_CONCAT	YES  이중 파이프 부호(    )를 문자열 병합 연산자로 사 용	NO  이중 파이프 부호(    )를 부울린 연산자 OR 로 사용

- 관련 문서: [데이터베이스 설정 > 구문/타입 파라미터](#)

## 새로 추가된 기능 - 연산자와 함수 관련

### CUBRIDSUS-3591 논리 연산자 확장 지원

부울린(Boolean) 연산식에 대한 논리 연산자를 확장 지원한다.

논리 연산자	비고
&&	AND 와 동일하게 사용 가능
	OR 와 동일하게 사용 가능 (단, PIPES_AS_CONCAT=NO 파라미터 설정 필요)
XOR	XOR 신규 지원
!	NOT 과 동일하게 사용 가능

```

csql> select * from code where s_name = 'X' && f_name = 'Mixed'
csql> ;x

=== <Result of SELECT Command in Line 1> ===

  s name          f name
=====
  'X'            'Mixed'

```

### CUBRIDSUS-3591 비교 연산자 확장 지원

비교 연산자를 확장 지원한다.

비교 연산자	비고
<=>	NULL Safe 등호 신규 지원
!=	<>와 동일하게 사용 가능
IS, IS NOT	조건식에서 연산식과 불리언 값(TRUE, FALSE, UNKNOWN, NULL)의 비교 연산 가능

- 관련 문서: [연산자와 함수 > 비교 연산자](#)

## CUBRIDSUS-3591 비트 연산자 및 비트 함수 신규 지원

비트열 리터럴을 확장하며, 비트 연산자 및 비트 연산 함수를 신규 지원한다.

지원 항목	비고
비트열 리터럴	2 진수 형식(B'1010', 0b1010) 및 16 진수 형식(X'a', 0xa) 사용 가능
비트 연산자	&,  , ^, ~, <<, >> 신규 지원
비트 함수	BIT_AND(expr), BIT_OR(expr), BIT_XOR(expr), BIT_COUNT(expr) 신규 지원

```
csql> select 17&3
csql> ;x

=== <Result of SELECT Command in Line 2> ===

              17&3
=====
              1

17 의 2 진수 →      10001
3 의 2 진수 → &(연산자) 00011
      값 →      00001
```

## CUBRIDSUS-3591 수치 연산 함수의 확장 지원

### ● 삼각 함수의 추가 지원

다양한 삼각 함수를 이용하여 라디안 값을 구할 수 있으며, 변환 함수를 이용하여 값의 단위를 라디안으로 변환 할 수도 있다. 이를 위해 새로 추가된 함수는 다음과 같다.

```
//삼각 함수를 이용하여 라디안 값을 반환
COS( X )
COT( X )
SIN( X )
TAN( X )

//역 삼각 함수를 이용하여 라디안 값을 반환
ACOS( X )
ASIN( X )
ATAN(X[, Y]), ATAN2(X[, Y])

//라디안 단위인 x를 각도로 환산하여 반환
DEGRESS( X )

// 각도 값 x를 라디안 단위 값으로 환산
RADIANS( X )

// PI 값을 DOUBLE로 반환
PI()
```

### ● 수학 함수의 추가 지원

절대 값, 거듭제곱 값, 제곱근 및 로그 연산을 위해 새로 추가된 함수는 다음과 같다.

```
//거듭 제곱 값 반환
POW( X )

//제곱근 반환
SQRT( X )

//이진 로그, 자연 로그, 상용 로그 값을 반환
LOG2( X ), LN( X ), LOG10( X )
```



- **SEED 값 지정이 가능하도록 랜덤 함수의 기능 확장**

CUBRID 가 지원하는 모든 랜덤 함수에서 seed 값을 지정할 수 있도록 기능이 확장되었다. 또한, 난수를 발생시키는 범위를  $0 \sim 2^{16}$  에서  $0 \sim 2^{32}$  로 확장하였다.

```
//정수 타입 난수 발생
RAND( [seed] )
RANDOM( [seed] )

//실수 타입 난수 발생
DRANDOM( [seed] )
DRAND( [seed] )
```

- **수치 값을 사용자가 지정한 형식의 문자열로 변환하는 FORMAT() 함수 추가 지원**

주어진 숫자를 '#,###,###.#####' 형식의 문자열로 변환하는 함수를 추가 지원하며, 소수부의 표현 자리 수를 인자로 지정할 수 있다.

```
// 주어진 값을 세 자리마다 쉼표로 구분하고 지정한 자릿수만큼 표현
SELECT FORMAT(12000.123456,4);

=====
'12,000.1235'
```

- 관련 문서: [수치 연산 함수](#) > [RANDOM/RAND 함수](#), [FORMAT 함수](#) 등

## CUBRIDSUS-3591 날짜/시간 함수의 확장 지원 및 출력 포맷의 다양화

- **날짜/시간 값에 대해 시간 간격 단위를 지정하여 연산을 수행하는 함수 추가 지원**

밀리초(MILLISECOND)부터 년(YEAR) 단위까지 다양한 시간 간격 단위를 지정하여 주어진 날짜 값에 특정 시간 간격과의 차이를 연산할 수 있다. 이를 위해 새로 추가된 함수는 다음과 같다.

```
//ADDDATE( ), DATE_ADD( ) 함수를 사용하여 주어진 날짜에서 시간 간격을 더한 값을 반환
SELECT ADDDATE('2010-06-20', INTERVAL 60 SECOND), ADDDATE('2010-06-20', 1);

=====
'12:01:00.000 AM 06/20/2010' '06/21/2010'

//SUBDATE( ), DATE_SUB( ) 함수를 사용하여 주어진 날짜에서 시간 간격을 뺀 값을 반환
SELECT SUBDATE('2010-06-20', INTERVAL 60 SECOND), SUBDATE('2010-06-20', 1);

=====
'11:59:00.000 PM 06/19/2010' '06/19/2010'

// 주어진 날짜 간 차이를 일 단위로 반환
SELECT DATEDIFF(SYSDATETIME, '2010-06-20');

=====
54
```

- **날짜/시간 값을 사용자가 지정한 형식의 문자열로 변환하는 함수의 추가 지원**

날짜/시간 값을 사용자가 지정한 형식의 문자열로 변환하는 함수를 추가 지원한다. 또한, 추가로 지원하는 '%'로 시작되는 새로운 포맷 지정자(specifier)를 결합하여 다양한 출력 포맷을 지정할 수 있다.

```
// 주어진 값을 'MM/DD/YYYY' 형식의 문자열을 출력
SELECT DATE(NOW());

=====
'08/13/2010'

// 주어진 값을 지정된 형식의 문자열로 변환 출력
SELECT DATE_FORMAT('2010-06-20 22:23:00', '%Y %M %W %H:%i'); ➔ 년/월/요일/시간:분
```

```
=====
'2010 June Sunday 22:23'

// time 값을 지정된 형식의 문자열로 변환 출력
SELECT TIME_FORMAT(sys_datetime, '%H %h %i %s %f'); ➔ 24 시/시/분/초/밀리초

=====
'00 12 13 34 566'
```

### ● 문자열을 날짜/시간 값으로 변환하는 함수의 추가 지원

문자열을 지정한 형식을 적용하여 날짜/시간 값으로 변환하는 함수를 추가로 지원한다. **STR\_TO\_DATE()** 함수에 적합한 포맷 지정자를 적용하여 문자열을 날짜/시간 값으로 변환할 수 있다. 또한, 문자열을 밀리초 단위의 타임스탬프 값으로 변환하는 **TIMESTAMP()** 함수를 추가로 제공한다.

```
// 문자열을 지정된 날짜/시간 타입으로 변환 출력
SELECT STR_TO_DATE('June 20, 2010', '%M %d, %Y');

=====
06-20-2010

// 문자열을 DATETIME 타입 값으로 반환 또는 시간 값을 더하여 출력
SELECT TIMESTAMP('2010-06-20'), TIMESTAMP('2010-06-20', '13:59:59');

=====
2010-06-20 00:00:00.000                2010-06-20 13:59:59.000
```

### ● 현재 시스템 날짜/시간을 구하는 함수의 추가 지원

시스템의 현재 날짜/시간을 구하는 기존 함수들의 동의어를 추가하였으며, UNIX 타임 스탬프 값을 출력하는 함수를 추가 지원한다.

```
// 현재 날짜를 반환
CURDATE(), CURRENT_DATE(), CURRENT_DATE, SYSDATE, SYS_DATE
➔ 2010-08-13

// 현재 시간을 반환
CURTIME(), CURRENT_TIME(), CURRENT_TIME, SYSTIME, SYS_TIME
➔ 00:28:48

// 현재 타임스탬프 값을 초 단위까지 반환
CURRENT_TIMESTAMP(), CURRENT_TIMESTAMP, LOCALTIME(), LOCALTIME, LOCALTIMESTAMP(), LOCALTIMESTAMP,
SYSTIMESTAMP, SYS_TIMESTAMP
➔ 2010-08-13 00:29:05

// 현재 타임스탬프 값을 밀리초 단위까지 반환
CURRENT_DATETIME(), CURRENT_DATETIME, NOW(), SYSDATETIME, SYS_DATETIME
➔ 2010-08-13 00:29:48.712

// UNIX_TIMESTAMP() 함수를 사용하여 '1970-01-01 00:00:00' UTC 이후 지정된 날짜까지의 초 단위 시간 간격을 반환
SELECT UNIX_TIMESTAMP('2010-06-20')

=====
1276959600
```

- 관련 문서: [날짜/시간 함수와 연산자](#), [ADDDATE 함수](#), [STR\\_TO\\_DATE 함수](#), [DATE\\_FORMAT 함수](#) 등

## CUBRIDSUS-3591 문자열 함수의 확장 지원

### ● 컬럼 값을 연결한 문자열을 가로 출력하는 CONCAT() 함수 지원

컬럼 값을 병합하여 **하나의 문자열로 출력**하는 **CONCAT()** 함수 및 연결할 값 사이에 구분자를 삽입하여 출력할 수 있는 **CONCAT\_WS()** 함수를 함께 제공한다.

```
SELECT CONCAT('CUBRID', '2008' , 'R3.0'), CONCAT_WS('/', 'CUBRID', '2008', 'R3.0');

=====
'CUBRID2008R3.0'      'CUBRID//2008//R3.0'
```

### ● 문자열 비교 및 검색하는 함수 추가 지원

주어진 위치로부터 특정 길이만큼의 문자열을 반환하는 함수로 **MID()** 함수를 추가로 지원하며, 가장 왼쪽 또는 가장 오른쪽 위치로부터 특정 길이만큼의 문자열을 반환하는 **LEFT()** 함수 및 **RIGHT()** 함수도 추가로 제공한다.

```
SELECT MID('12345abcde', 6, 4); → 6 번째문자 시작위치포함 4 문자
SELECT LEFT('12345abcde', 4); → 앞에서 4 문자
SELECT RIGHT('12345abcde', 4); → 끝에서 4 문자

=====
'abcd'      '1234'      'bcde'
```

주어진 문자열의 위치를 반환하는 함수로 **LOCATE()** 함수를 추가로 제공한다.

```
SELECT LOCATE ('abc', '12345abcde'); → 시작위치

=====
6
```

주어진 문자열과 나머지 인자로 지정된 문자열을 비교하여, 동일한 인자의 위치를 반환하는 **FIELD()** 함수를 추가로 제공한다.

```
SELECT FIELD('abc', 'a', 'ab', 'abc', 'abcd'); → 첫 인자와 동일한 문자열비교 하여 인자의 위치 반환

=====
3
```

주어진 두 개의 문자열을 비교하여 0, 1, -1 을 반환하는 **STRCMP()** 함수를 추가로 제공한다.

```
SELECT STRCMP('abc', 'abc'); → 첫 인자와 동일 0
SELECT STRCMP('abc', 'ab'); → 첫 인자가 크면 1
SELECT STRCMP('abc', 'abcd'); → 첫 인자가 작으면 -1
SELECT STRCMP('abc', 'ac'); → 문자열이 같지 않으면 -1
SELECT STRCMP('abc', NULL); → 인자 중 NULL 이 있으면 NULL
```

### ● 문자열을 변환하는 함수 추가 지원

주어진 문자열을 대문자 또는 소문자로 변환하는 **UCASE()** 함수 및 **LCASE()** 함수를 각각 추가 지원한다. 또한, 주어진 문자열을 역순으로 변환하여 반환하는 **REVERSE()** 함수를 추가 지원한다.

```
SELECT UCASE('CUBRID'), LCASE('CUBRID'), REVERSE('CUBRID');
⇨ UPPER(), LOWER() 동일

=====
'CUBRID'      'cubrid'      'dirBUC'
```

- 관련 문서: [스트링 함수와 연산자](#)> [CONCAT 함수](#), [MID 함수](#), [LOCATE 함수](#), [REVERSE 함수](#) 등

## CUBRIDSUS-3591 정보 함수의 확장 지원

### ● 데이터베이스 관련 정보를 반환하는 함수 추가 지원

현재 연결된 데이터베이스 이름, 대상 서버에 존재하는 모든 데이터베이스 이름 및 데이터베이스 사용자 이름을 반환하는 정보 함수를 추가 지원한다.

```
//연결된 데이터베이스 이름 반환
SELECT DATABASE(), SCHEMA();

=====
demodb@localhost      demodb@localhost

//서버에 존재하는 모든 데이터베이스 이름 반환
SELECT LIST_DBS();

=====
```

```
testdb demodb

//데이터베이스 사용자 이름 및 호스트 이름을 반환
SELECT USER(), SYSTEM_USER();
=====
DBA@janus-PC      DBA@janus-PC
```

- 컬럼에 정의된 디폴트 값을 반환하는 함수 추가 지원

주어진 컬럼에 정의된 디폴트 값을 반환하는 **DEFAULT()** 함수를 추가 지원한다. [질의로 DEFAULT 값을 확인](#)

```
CREATE TABLE info_tbl(id INT DEFAULT 0)
INSERT INTO info_tbl VALUES (1);

//컬럼에 정의된 디폴트 값 반환
SELECT id, DEFAULT(id) FROM info_tbl;

=====
1          0
```

- 이전 질의문에 의해 변경된 행의 개수를 반환하는 함수 추가 지원

이전 질의문에 의해 변경(UPDATE, INSERT, DELETE)된 행의 개수를 반환하는 함수를 추가로 지원한다.

[SQL 이 생성된 클라이언트 세션에 한정\(csql 에 한정\)](#)

```
INSERT INTO info_tbl VALUES (4), (5), (6);
//변경된 행의 개수를 반환
SELECT ROW_COUNT();
;xr

=====
3
```

- 관련 문서: [정보 함수 > USER 함수](#), [DEFAULT 함수](#), [ROW\\_COUNT 함수](#) 등

## CUBRIDSUS-3591 조건 연산 함수의 확장 지원

- NULL 값 비교를 위한 ISNULL(), IFNULL() 함수 추가 지원

특정 컬럼 값이 NULL 인지 비교하고, 결과에 따라 다른 값을 반환하는 함수를 추가 지원한다.

```
CREATE TABLE case_tbl(a INT);
INSERT INTO case_tbl VALUES (1), (2), (NULL);
//expr 이 NULL 이면 1, NULL 이 아니면 0 을 반환
SELECT ISNULL(a) FROM case_tbl; ➔ 정수형

isnull(a)
=====
0
0
1

//expr1 이 NULL 이면 expr2, NULL 이 아니면 expr1 을 반환 ➔ 문자형
SELECT IFNULL(a, 'UNKNOWN') FROM case_tbl;

ifnull(a, 'UNKNOWN')
=====
'1'
'2'
'UNKNOWN'
```

- 인자 값 비교를 위한 IF(), NULLIF() 함수 추가 지원

인자 값의 참/거짓 또는 인자 간 동등 비교를 수행하고, 결과에 따라 다른 값을 반환하는 함수를 추가 지원한다.

```
//expr1 이 참이면 expr2 를, 거짓이면 expr3 을 반환
SELECT IF(a=1, 'one', 'other') FROM case_tbl;
```

```

if(a=1, 'one', 'other')
=====
'one'
'other'
'other'

//expr1 과 expr2 가 동일하면 NULL 을 반환, 다른면 expr1 을 반환,
SELECT NULLIF(a, 1) FROM case_tbl;

      a  nullif(a, 1)
=====
      1          NULL
      2           2
     NULL        NULL

```

- 관련 문서: [조건 연산식과 함수](#) > [IFNULL 함수](#), [NULLIF 함수](#), [IF 함수](#) 등

## 새로 추가된 기능 - 기타

### CUBRIDSUS-3005, 3085 데이터베이스 구동 중에도 공간 정리 작업을 수행할 수 있는 compactdb 유틸리티 옵션 지원

이전 버전에서는 데이터베이스 프로세스가 정지 상태인 경우에만 **cubrid compactdb** 유틸리티를 실행하여 공간 정리 작업을 수행할 수 있었으나, 구동 중인 상태(online)에서도 이를 수행할 수 있도록 클라이언트/서버 모드 실행 옵션(-C)을 추가 지원한다. 또한, 클라이언트/서버 모드에서만 적용할 수 있는 상세 옵션(-I, -i, -c, -d, -p)을 추가로 지원한다.

```
usage: cubrid compactdb [OPTION] database-name [class_name1, class_name2,...]
-i, --input-class-file=FILE : 이 옵션을 사용하여 대상 테이블 이름을 포함하는 입력 파일 이름을 지정할 수 있다. 라인 당 하나의 테이블 이름을 명시하며, 유효하지 않은 테이블 이름은 무시된다. 이 옵션을 지정하는 경우, 데이터베이스 이름 뒤에 대상 테이블 이름 리스트를 직접 명시할 수 없으므로 주의한다.
-p, --pages-committed-once=NUMBER : 이 옵션을 사용하여 한 번에 커밋 할 수 있는 최대 페이지 수를 지정할 수 있다. 디폴트 값은 10 이며, 최소 값은 1, 최대 값은 10 이다. 옵션 값이 작으면 클래스/인스턴스에 대한 잠금 비용이 작으므로 동시성은 향상될 수 있으나 작업 속도는 저하될 수 있고, 옵션 값이 크면 동시성은 저하되나 작업 속도는 향상될 수 있다.
-d, --delete-old-repr : 이 옵션을 사용하여 카탈로그에서 과거 테이블 표현을 삭제할 수 있다.
-I, --Instance-lock-timeout : 이 옵션을 사용하여 인스턴스 잠금 타임아웃 값을 지정할 수 있다. 디폴트 값은 2(초)이며, 최소 값은 1, 최대 값은 10 이다. 설정된 시간동안 잠금 인스턴스를 대기하므로, 옵션 값이 작을수록 작업 속도는 향상될 수 있으나 처리 가능한 인스턴스 개수가 적어진다. 반면, 옵션 값이 클수록 작업 속도는 저하되나 더 많은 인스턴스에 대해 작업을 수행할 수 있다.
-c, --class-lock-timeout : 이 옵션을 사용하여 클래스 잠금 타임아웃 값을 지정할 수 있다. 디폴트 값은 10(초)이며, 최소값은 1, 최대 값은 10 이다. 설정된 시간동안 잠금 테이블을 대기하므로, 옵션 값이 작을수록 작업 속도는 향상될 수 있으나 처리 가능한 테이블 개수가 적어진다. 반면, 옵션 값이 클수록 작업 속도는 저하되나 더 많은 테이블에 대해 작업을 수행할 수 있다.
```

### CUBRIDSUS-2923, 2961, 2972 PHP API에 20여 개의 함수 추가 지원

PHP API에 fetch 관련, field 정보 관련, DB 정보 관련 함수들을 추가로 지원한다.

Fetch 관련 함수	Field 관련 함수	DB 정보 관련 함수	기타 함수
cubrid_fetch_assoc( )	cubrid_field_flags( )	cubrid_get_client_info( )	cubrid_data_seek( )
cubrid_fetch_field( )	cubrid_field_len( )	cubrid_get_server_info( )	cubrid_free_result( )
cubrid_fetch_lengths( )	cubrid_field_name( )	cubrid_get_charset( )	cubrid_insert_id( )
cubrid_fetch_object( )	cubrid_field_seek( )	cubrid_get_db_parameter( )	cubrid_result( )
cubrid_fetch_row( )	cubrid_field_table( )	cubrid_list_dbs( )	cubrid_unbuffered_query( )
	cubrid_field_type( )		cubrid_real_escape_string( )

---

cubrid\_num\_fields( )

---

- 관련 문서: [PHP API](#)

---

## CUBRIDSUS-2829 insert\_execution\_mode 파라미터에서 설정할 수 있는

### INSERT모드 추가

**INSERT** 문법 확장 및 **REPLACE** 문 지원으로 인해 **INSERT** 실행 모드가 추가되어, 총 5 개의 실행 모드의 조합으로 **insert\_execution\_mode** 파라미터 값을 설정할 수 있다. 이전 버전에서 지원된 **INSERT** 실행 모드는 INSERT\_SELECT, INSERT\_VALUES, INSERT\_DEFAULT였으며, INSERT\_REPLACE, INSERT\_ON\_DUP\_KEY\_UPDATE가 새로 추가되었다.

- 관련 문서: [데이터베이스 서버 설정 > insert\\_execution\\_mode](#)