12장. 데이터 정의어(DDL)

DDL은 Data Definition Language의 약자로 데이터 정의어를 말합니다. 즉, 데이터베이스를 정의하는 언어를 말하며 데이터를 생성하거나 수정, 삭제 등 데이터의 전체 골격을 결정하는 역할을 맡고있습니다. 데이터 정의어에는 CREATE, ALTER, DROP, TRUNCATE가 있습니다. DDL의 대상은 테이블 뿐만 아니라, 데이터베이스(또는 스키마), 인덱스, 뷰 등이 될 수 있지만 관계형 데이터베이스에 데이터가 저장되는 기본 구조인 테이블 기준으로 설명드립니다.

12.1 CREATE TABLE

12.1.1 Syntax

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } | UNLOGGED ] TABLE [ IF NOT EXISTS ]
table_name ( [
  { column_name data_type [ COLLATE collation ] [ column_constraint [ ... ] ]
    | table_constraint
    | LIKE source_table [ like_option ... ] }
    [, ... ]
] )
```

(더 많은 옵션이 존재하나, 자주 사용되는 부분만 기재)

관계형 데이터베이스의 테이블은 종이에 작성된 테이블과 매우 유사합니다. 테이블은 행과 열로 구성됩니다. 열의 수와 순서는 고정되어 있으며 각 열의 이름이 지정됩니다. 행의 수는 가변적입니다. 즉, 행 수는 그 시점에서 얼마나 많은 데이터가 저장되는지를 나타냅니다. SQL은 테이블의 행 순서를 보장하지 않습니다. 테이블을 읽으면 명시적으로 정렬이 요구되지 않는 한 행은 불특정 순서로 반환됩니다. 또한 SQL은 행에 고유한 식별자를 할당하지 않으므로 테이블에 정확히 동일한 행이 여러 개 있을 수 있습니다.

CREATE TABLE은 현재 데이터베이스에 데이터가 비어 있는 새 테이블을 만듭니다. 테이블은 명령을 실행하는 사용자가 소유합니다. Schema 이름이 지정되면(예: CREATE TABLE myschema.mytable ...) 지정된 Schema에 테이블이 생성되고 그렇지 않으면 현재 Schema에 생성됩니다. 임시 테이블은 특수한 Schema에 존재하므로 임시 테이블을 생성할 때 Schema 이름을 지정할 수 없습니다. 테이블의 이름은 동일한 Schema에 있는 다른 테이블, 시퀀스, 인덱스, 뷰 또는 외부 테이블의 이름과 구별되어야 합니다. 위의 varchar(80)은 최대 80자의 문자열을 저장할 수 있는 데이터 형식을 지정합니다. int 는 일반적인 정수용의 형태이고 real은 단정밀도 부동 소수점 숫자를 저장하는 형식입니다. date는 이름에서 알 수 있듯이 날짜입니다.

AgensSQL은 표준 SQL의 데이터형, int, smallint, real, double precision, char(N), varchar(N), date, time, timestamp나 interval을 지원합니다. 또한 일반적인 유틸리티를 위한 유형과 고급 기하 데이터 유형도 지원합니다.

12.1.2 데이터 타입

각 열에는 데이터 형식이 있습니다. 데이터 유형은 열에 할당되는 값을 제한합니다. 또한 열에 저장된 데이터에 의미가 할당되어 데 이터를 계산에 사용할 수 있습니다. 예를 들어, 수치형과 선언된 열은 임의의 텍스트 캐릭터 라인은 받아들이지 않습니다. 그런 다음 숫자 형식의 열에 저장된 데이터를 산술 계산에 사용할 수 있습니다. 대조적으로 문자열 유형으로 선언 된 열은 거의 모든 종류 의 데이터를 허용합니다. 그러나 문자열 조인과 같은 연산에는 사용할 수 있지만 산술 계산에는 사용할 수 없습니다. 자주 사용되는 데이터형으로는, 정수를 나타내는 integer, 소수도 나타낼 수가 있는numeric, 캐릭터 라인을 나타내는 text, 일자를 나타내는 date, 시각을 나타내는 time, 그리고 일자와 시각의 양쪽 모두를 포함한 timestamp가 있습니다.

아래의 표에 PostgreSQL에 내장된 범용 데이터 유형을 모두 보여줍니다. " 별칭 " 열에 나열된 이름은 PostgreSQL 에서 내부적으로 사용된 이름입니다 . 또한 일부 내부적으로 사용되거나 더 이상 사용되지 않는 유형을 사용할 수 있지만 여기에 나열되지 않았습니다.

이름	별칭	설명
bigint	int8	부호 있는 8바이트 정수
bigserial	serial8	자동 증가 8바이트 정수
bit [(n)]		고정 길이 비트 문자열
bit varying [(n)]	varbit [(n)]	가변 길이 비트 문자열
boolean	bool	논리 부울(참/거짓)
box		평면 위의 직사각형 상자
bytea		이진 데이터(" 바이트 배열 ")
character [(n)]	char [(n)]	고정 길이 문자열
character varying [(n)]	varchar [(n)]	가변 길이 문자열
cidr		IPv4 또는 IPv6 네트워크 주소
circle		비행기의 원
date		달력 날짜(년, 월, 일)
double precision	float8	배정밀도 부동 소수점 숫자(8바이트)
inet		IPv4 또는 IPv6 호스트 주소
integer	int,int4	부호 있는 4바이트 정수
interval [fields] [(p)]		시간 범위
json		텍스트 JSON 데이터

jsonb		이진 JSON 데이터, 분해
line		비행기 위의 무한선
lseg		평면의 선분
macaddr		MAC(미디어 액세스 제어) 주소
macaddr8		MAC(Media Access Control) 주소(EUI-64 형식)
money		통화 금액
numeric [(p, s)]	decimal [(p, s)]	선택 가능한 정밀도의 정확한 숫자
path		평면의 기하학적 경로
pg_lsn		PostgreSQL 로그 시퀀스 번호
pg_snapshot		사용자 수준 트랜잭션 ID 스냅샷
point		평면 위의 기하학적 점
polygon		평면에서 닫힌 기하학적 경로
real	float4	단정밀도 부동 소수점 숫자(4바이트)
smallint	int2	부호 있는 2바이트 정수
smallserial	serial2	자동 증가 2바이트 정수
serial	serial4	자동 증가 4바이트 정수
text		가변 길이 문자열
time [(p)] [without time zone]		시간(시간대 없음)
time [(p)] with time zone	timetz	시간대를 포함한 하루 중 시간
timestamp [(p)] [without time zone]		날짜 및 시간(시간대 없음)
timestamp [(p)] with time zone	timestamptz	시간대를 포함한 날짜 및 시간
tsquery		텍스트 검색 쿼리
tsvector		텍스트 검색 문서
txid_snapshot		사용자 수준 트랜잭션 ID 스냅샷(더 이상 사용되지 않음, 참조 pg_snapshot)
uuid		보편적으로 고유한 식별자
xml		XML 데이터

12.1.3 Default

열에 기본값을 지정할 수 있습니다. 새로 작성된 행의 일부 열에 값이 지정되어 있지 않은 경우, 이러한 공란에 각 열의 기본값으로 채워집니다. 데이터 조작 명령을 사용하여 열을 기본값으로 설정하도록 명시 적으로 요청할 수도 있습니다. 명시적으로 선언된 기본값이 없으면 기본값은 NULL 값입니다. NULL값은 알 수 없는 데이터를 나타내는 것이며 일반적으로 이 방법으로 문제가 되지 않습니다. 테이블 정의에서 기본값은 열 데이터 형식 뒤에 열거됩니다.

```
CREATE TABLE products
   ( product_no integer,
     name text,
     price numeric DEFAULT 9.99
    );

   Column   |    Type   | Collation | Nullable | Default
---------------+------------+------------+-----------+---------
 product_no | integer   |            |            |
 name         | text       |           |            |
 price        | numeric |           |            |  9.99
```

기본값을 표현식으로 사용할 수 있으며, 기본값이 삽입될 때마다 적용됩니다. 흔한 예로, timestamp 열이 삽입시간으로 설정되도록 열은 기본 CURRENT_TIMESTAMP를 가질 수 있습니다. 또 다른 예는 각 행에 "번호"를 할당하는 경우입니다. PostgreSQL에서는, 일반적으로 이하와 같이 기술하는것으로 생성됩니다.

```
CREATE TABLE products
   ( product_no SERIAL,
   … );
```

12.1.4 Constraint

Constraint은 가장 범용적인 제약 유형입니다. 이를 사용하여 특정 컬럼의 값이 논리 값의 표현식을 충족하도록(실제 값) 지정 할 수 있습니다. 예를 들어, 제품 가격을 반드시 양수로 만들려면 다음을 수행하십시오.

```
CREATE TABLE products
   ( product_no integer,
```

```
    name text,
    price numeric CHECK (price > 0)
);
```

따라서 제약 조건 정의는 기본값 정의와 마찬가지로 데이터 유형 뒤에 옵니다. 기본값과 제약 조건은 임의의 순서로 열거할 수 있습니다. 검사 제약은 CHECK 키워드 뒤에 오는 괄호로 묶인 표현식으로 구성됩니다. 검사 제약 조건 표현식에는 제한된 열이 포함되어야 합니다.

제약 조건에 개별적으로 이름을 지정할 수도 있습니다. 이름을 지정하면 오류 메시지를 쉽게 이해할 수있으며 변경하려는 제약 조건을 볼 수 있습니다. 구문은 다음과 같습니다.

```
CREATE TABLE products
      ( product_no integer,
        name text,
        price numeric CONSTRAINT positive_price CHECK (price > 0)
       );
```

위에서 언급했듯이 명명된 제약 조건은 CONSTRAINT 키워드로 시작하여 식별자, 제약 조건 정의를 따릅니다. (이 방법으로 제약 이름을 지정하지 않으면 시스템에서 이름을 지정합니다.)

검사 제약 조건에서 여러 열을 참조할 수도 있습니다. 예를 들어 일반 가격과 할인 가격을 저장할 때 할인 가격이 일반 가격보다 낮아야 합니다.

```
CREATE TABLE products
      ( product_no integer,
        name text,
        price numeric CHECK (price > 0),
        discounted_price numeric CHECK (discounted_price > 0),
        CHECK (price > discounted_price)
       );
```

세 번째 제약은 새로운 구문을 사용합니다. 이것은 특정 열에 추가되지 않고 쉼 표로 구분된 열 목록의 별도 항목으로 나타납니다. 열 정의 및 이러한 제약 정의는 임의의 순서로 열거할 수 있습니다.

처음 두 개의 제약을 열 제약이라고합니다. 반대로 세 번째 제약 조건은 열 정의와 별도로 작성되므로 테이블 제약 조건이라고합니 다. 열 제약 조건을 테이블 제약 조건으로 작성할 수는

있지만 그 반대는 가능하거나 불가능할 수 있습니다. 왜냐하면 열 제약 조건은 제약 조건과 연관된 열만 참조하기 때문입니다.

위의 예는 다음과 같이 쓸 수도 있습니다.

```
CREATE TABLEproducts
    ( product_no integer,
      name text,
      price numeric,
      CHECK (price > 0),
      discounted_price numeric,
      CHECK (discounted_price > 0),
      CHECK (price > discounted_price)
     );
```

또는 다음과 같이 사용할 수 있습니다.

```
CREATE TABLEproducts
    ( product_no integer,
      name text,
      price numeric CHECK (price > 0),
      discounted_price numeric,
      CHECK (discounted_price > 0 AND price > discounted_price)
     );
```

어떤 방법을 사용해도 무관합니다.

열 제약 조건과 마찬가지로 테이블 제약 조건에 이름을 지정할 수 있습니다.

```
CREATE TABLEproducts
    ( product_no integer,
      name text,
      price numeric,
      CHECK (price > 0),
      discounted_price numeric,
      CHECK (discounted_price > 0),
      CONSTRAINT valid_discount CHECK (price > discounted_price)  );
```

검사 제약 조건에서 검사 표현식이 참 또는 널값으로 평가되면 조건이 충족된다는 점에 유의하십시오. 대부분의 표현식은 하나의 연산항목에 대해 널(NULL)이 있으면 널(NULL)로 평가됩니다. 제약 조건은 제약 대상 열에 널(NULL) 값이 포함되는 것을 방지하지 않습니다.

컬럼이 널(NULL) 값을 포함하지 않도록 하기 위해 다음절에서 설명하는 널(null) 제약조건을 사용할 수 있습니다.

12.1.5 NOT NULL Constraint

NOT NULL 제약 조건은 열이 NULL 값을 가질 수 없도록 지정합니다. 구문의 예는 다음과 같습니다.

```
CREATE TABLEproducts
    ( product_no integer NOT NULL,
      name text NOT NULL,
      price numeric );
```

NOT NULL 제약 조건은 항상 열 제약 조건으로 설명됩니다. NOT NULL 제약은 CHECK (column_name IS NOT NULL)라는 검사 제약과 기능적으로는 동등하지만, PostgreSQL에서는 명시적으로 NOT NULL 제약을 작성하는 것이 보다 효과적입니다. 이와 같이 작성된 NOT NULL 제약에 명시적인 이름을 붙일 수 없는 것이 단점입니다.

하나의 열에 여러 제약 조건을 적용할 수도 있습니다. 그렇게하려면, 차례차례로 제약을 기입합니다.

```
CREATE TABLEproducts
    ( product_no integer NOT NULL,
      name text NOT NULL,
      price numeric NOT NULL CHECK (price > 0)  );
```

12.1.6 UNIQUE Constraint

고유성 제약 조건은 컬럼 또는 컬럼 그룹에 포함된 데이터가 테이블의 모든 행에서 고유한지 확인합니다. 열 제약의 경우 구문은 다음과 같습니다.

```
CREATE TABLE products
    ( product_no integer UNIQUE,
      name text,
      price numeric );
```

또한 테이블 제약조건의 구문은 다음과 같습니다.

```
CREATE TABLE products
```

```
    ( product_no integer,
      name text,
      price numeric,
      UNIQUE(product_no)  );
```

컬럼 세트에 대해 고유성 제약 조건을 정의하려면 컬럼 이름을 쉼표로 구분하여 테이블 제약 조건으로 작성하십시오.

```
CREATE TABLE example
    ( a integer,
      b integer,
      c integer,
      UNIQUE(a, c));
```

지정된 열의 값 조합이 전체 테이블에서 고유하다는 것을 지정합니다. 반드시 열 중 하나가 고유할 필요는 없습니다 (일반적으로 고유하지않음).
고유성 제약에는, 통상의 방법으로 이름을 할당할 수도 있습니다.

```
CREATE TABLE products
    ( product_no integer CONSTRAINT must_be_different UNIQUE,
      name text,
      price numeric );
```

고유성 제약 조건을 추가하면 제한조건에 지정된 열 또는 열 그룹에 대해 고유한 B-트리 색인이 자동으로 만들어집니다. 일부 행에만 적용되는 고유성 제한을 고유성 제약조건으로 작성할수는 없지만, 이러한 제한을 고유한 부분 색인을 작성하여 실현할 수 있습니다.

일반적으로 제약 조건이 적용되는 모든 열에 대해 동일한 값을 갖는 행이 테이블에 두 개 이상의 행을 갖는 경우 고유한 제약 조건 위반이 발생합니다. 그러나 이 비교에서 두 개의 NULL값은 결코 같은 값으로 간주되지 않습니다. 즉, 고유한 제약 조건이 있더라도 제약 조건이 있는 열 중 적어도 하나에 NULL값이 있는 여러 행을 저장할 수 있습니다. 이 동작은 표준 SQL을 준수하지만 이 규칙을 따르지 않는 SQL 데이터베이스가 있으므로 응용프로그램을 개발할 때는주의해야 합니다.

12.1.7 PRIMARY KEY

기본 키 제약 조건은 컬럼 또는 컬럼 그룹이 테이블의 행을 고유하게 식별하는 것으로 사용할 수 있음을 의미합니다. 이렇게하려면 값이 고유하고 NULL값이 아니어야 합니다. 즉, 다음 두 테이블 정의는 동일한 데이터를 허용합니다.

```
CREATE TABLE products
    ( product_no integer UNIQUE NOT NULL,
     name text,
     price numeric);
```

```
CREATE TABLE products
    ( product_no integer PRIMARY KEY,
     name text,
     price numeric );
```

기본 키도 여러 열을 설정할 수 있으며 구문은 고유성 제약 조건과 유사합니다.

```
CREATE TABLE example
    ( a integer,
     b integer,
     c integer,
     PRIMARY KEY(a, c)  );
```

기본 키를 추가하면 기본 키에 지정된 열 또는 열 그룹에 대해 고유한 B-트리 색인이 자동으로 만들어집니다. 또한 열에 NOT NULL 표시가 적용됩니다.

하나의 테이블은 최대 하나의 기본 키를 가질 수 있습니다. 관계형 데이터베이스 이론에서는, 모든 테이블에 기본 키가 하나 필요합니다. 이 규칙은 PostgreSQL에서는 강제되지 않지만 대부분의 경우에는 이것을 따르는 것이 좋습니다.

기본 키는 문서화 및 클라이언트 응용 프로그램 모두에서 유용합니다. 예를 들어, 행 값을 변경할 수 있는 GUI 응용프로그램이 행 을 고유하게 식별하려면 아마 테이블의 기본 키를 알아야 합니다. 다른 기본 키가 선언 될 때 데이터베이스 시스템이 이를 사용하는몇 가지 장면이 있습니다. 예를 들어, 외래 키가 테이블을 참조하면 기본 키가 기본 대상 열입니다.

12.1.8 FOREIGN KEY

외래 키 제약 조건은 열 (또는 열 그룹)의 값이 다른 테이블의 행 값과 일치해야 함을 지정합니다. 이렇게 하면 연관된 두 테이블 의 참조 무결성이 유지됩니다.

앞에서 예로 사용했던 products 테이블을 생각해 보겠습니다.

```
CREATE TABLE products
    ( product_no integer PRIMARY KEY,
      name text,
      price numeric );
```

또한 이러한 제품에 대한 주문을 저장하는 테이블을 작성했다고 가정합니다. 이 주문의 주문 테이블에는 실제로 존재하는 제품의 주문만 저장하고 싶습니다. 따라서 products 테이블을 참조하는 orders 테이블에 외래키 제약 조건을 정의합니다.

```
CREATE TABLE orders
    ( order_id integer PRIMARY KEY,
      product_no integer REFERENCES products (product_no),
      quantity integer  );
```

이제 products 테이블에 존재하지 않는 NULL이 아닌 product_no 항목을 사용하여 주문을 작성할 수 없습니다.

이 경우 orders 테이블을 참조 테이블, product 테이블을 기준 테이블이라고 합니다. 마찬가지로 참조 열과 기준 열도있습니다.

위의 명령은 다음과 같이 줄일 수 있습니다.

```
CREATE TABLE orders
    ( order_id integer PRIMARY KEY,
      product_no integer REFERENCES products,
      quantity integer  );
```

열 목록이 없기 때문에 참조 테이블의 기본 키가 참조 열로 사용됩니다.

외래 키에서도 열 그룹을 제한하거나 참조할 수있습니다. 이것도 테이블 제약의 형태로 기술할 필요가 있습니다.

```
CREATE TABLE t1
    ( a integer PRIMARY KEY,
```

```
    b integer,
    c integer,
    FOREIGN KEY(b, c) REFERENCES other_table (c1, c2) );
```

물론 제한된 열 수와 유형은 참조된 열의 수와 유형이 일치해야합니다.

외래 키 제약에는, 통상의 방법으로 이름을 할당할 수도 있습니다. 테이블은 여러 개의 외래 키 제약 조건을 가질 수 있습니다. 이것은 테이블 간의 다 대 다 관계를 구현하는 데 사용됩니다.

예를 들어, 제품 및 주문에 대한 각 테이블이 있는 경우 여러 제품에 걸쳐있는 주문을 가능하게 하려는 경우(위의예의구조에서는불가능함) 다음 테이블 구조를 사용할 수 있습니다.

```
CREATE TABLE products
    ( product_no integer PRIMARY KEY,
     name text,
     price numeric
    );

CREATE TABLE orders
    ( order_id integer PRIMARY KEY,
      shipping_address text,
      ...
    );

CREATE TABLE order_items
    ( product_no integer REFERENCES products,
     order_id integer REFERENCES orders,
     quantity integer,
     PRIMARY KEY(product_no, order_id)  );
```

마지막 테이블에서 기본 키와 외래 키가 겹치는 것에 주목하십시오.

외래 키가 제품과 연관되지 않은 주문을 작성할 수 없는 것은 이미 설명한 것입니다. 그러나 한 주문에서 참조한 제품이 주문 후 삭제되면 어떻게 될 것입니다. SQL 에서는 이러한 경우도 취급할 수가 있습니다. 직관적으로 몇 가지 옵션이 있습니다.

- 참조된 항목을 삭제할 수 없음
- 기본 키도 함께 삭제

구체적인 예로서, 위의 예제의 다 대다 관계에 다음 정책을 구현해 봅시다. (order_items를 통해) 주문에서 참조된 상태 로 제품을 삭제하려고 하면 이 작업을 수행할 수 없습니다. 주문이 삭제되면 주문 항목도 삭제됩니다.

```
CREATE TABLE products
    ( product_no integer PRIMARY KEY,
      name text,
      price numeric
    );

CREATE TABLE orders
    ( order_id integer PRIMARY KEY,
      shipping_address text,
      ...
    );

CREATE TABLE order_items
    ( product_no integer REFERENCES products ON DELETE RESTRICT,
      order_id integer REFERENCES orders ON DELETE CASCADE,
      quantity integer,
      PRIMARY KEY(product_no, order_id)
      );
```

DELETE RESTRICT와 DELETE CASCADE 라는 두 가지는 가장 일반적인 옵션입니다. RESTRICT는 참조된 행이 삭제되는 것을 방지합니다. NO ACTION은 제약 조건이 검사될 때 참조행이 여전히 존재하는 경우 오류가 있음을 의미합니다. 이것은 아무것도 지정하지 않은 경우의 기본 동작입니다. CASCADE는 참조된 행이 삭제될 때 참조할 행도 삭제되도록 지정합니다. 다른 두 가지 옵션, SET NULL 및 SET DEFAULT가 있습니다. 이들은 참조 행이 삭제될 때 참조행의 참조 열이 각각 널 또는 각 열의 기본값으로 설정됩니다. 이것들은 제약을 지키는 것을 면제하지 않는다는점에 유의하십시오. 예를 들어, 조작에 SET DEFAULT를 지정해도 기본값이 외래키 제약 조건을 충족시키지 않으면 조작이 실패합니다.

ON DELETE와 비슷하며 피 참조열이 변경(업데이트)되었을 때 호출되는 ON UPDATE도있습니다. 이것들이 할 수 있는 액션은 같습니다. 이 경우 CASCADE는 참조된 열의 업데이트된 값이 참조 행에 복사됩니다.

통상적으로 참조 행은 그 참조열의 어느쪽이든 NULL 의 경우는 외래키 제약을 채울 필요가 없습니다. MATCH FULL이 외래키 선언에 추가되면 참조열이 모두 null 인경우에만 참조행이 제약 조건을 충족하는 것을 피할 수 있습니다 (즉, NULL과 NULL이 아닌 조합은 MATCH FULL 제약 조건에 위반이 보장됩니다). 참조 행이 외래키 제약 조건을 충족시키지 못할 가능성을 제거하려면 참조열을 NOT NULL로 선언하십시오.

외래 키는 기본 키이거나 고유성 제약 조건을 구성하는 열을 참조해야합니다. 이는 참조가 항상(기본 키 또는 고유 제약 조건의 기초가 됨) 인덱스를 가짐을 의미합니다. 따라서 참조 행과 일치하는 행이 있는지 확인하는 것이 효율적입니다. 참조된 테이블의 행 DELETE 또는 참조된 행의 UPDATE는 이전 값과 일치하는 행에 대한 참조 테이블을 스캔해야하므로 참조 행에 인덱스를 추가하는 것이 일반적으로 좋은 생각입니다. 이것은 항상 필요한 것은 아니며, 인덱스의 방법에는 많은 선택사항이 있으므로, 외래 키 제약의 선언에서는 참조 열의 인덱스가 자동적으로 만들어지는 것은 아닙니다.

12.1.9 시스템 열

모든 테이블에는 시스템에 의해 정의된 여러 시스템 열이 있습니다. 따라서 시스템 열의 이름을 사용자 정의 열의 이름으로 사용할 수 없습니다.

tableoid

이 행을 포함하는 테이블의 OID입니다. 이 열은 특히 상속 계층 구조에서 선택 쿼리에 유용합니다 (5.10 참조). 이 열이 없으면 어떤 테이블에서 그 행이 왔는지 알기 어렵기 때문입니다. tableoid를 pg_class의 oid 열에 조인하여 테이블 이름을 얻을 수 있습니다.

xmin

이 행 버전의 삽입 트랜잭션의 식별 정보(트랜잭션 ID)입니다. (행 버전은 행의 개별 상태입니다. 행이 업데이트 될 때마다 동일한 논리 행에 대한 새 행 버전이 만들어집니다.)

cmin

삽입 트랜잭션(0부터 시작)의 명령 식별자입니다.

xmax

삭제 트랜잭션의 식별 정보(트랜잭션 ID)입니다. 삭제되지 않은 행 버전에서는 0입니다. 보이는 행 버전에서 이 열이 0이 아닌 경우가 있습니다. 이것은 일반적으로 삭제 트랜잭션이 아직 커밋되지 않았거나 삭제 시도가 롤백되었음을 의미합니다.

cmax

삭제 트랜잭션 내의 명령 식별자 또는 0입니다.

ctid

테이블 내의 행 버전의 물리적 위치를 나타냅니다. ctid는 행 버전을 신속하게 찾을 수 있지만 행 ctid는 VACUUM FULL로 업데이트하거나 이동할 때 변경됩니다. 따라서 ctid는 장기 행 식별자로 사용할 수 없습니다. 논리 행을 식별하려면 기본 키를 사용해야 합니다.

12.1.10 Inherit

PostgreSQL은 데이터베이스 설계자에게 편리한 테이블 상속을 구현합니다.
도시(cities)의 데이터 모델을 만들려고 한다고 가정합니다. 각 주에는 많은 도시가 있지만, 수도(capitals)는 1개입니다. 어느 주에 대해서도 수도를 신속하게 검색하고 싶습니다. 이것은 두 개의 테이블을 작성하여 얻을 수 있습니다. 하나는 수도의 테이블이고 다른 하나는 수도가 아닌 도시의 테이블입니다. 그러나 수도인지 여부에 관계없이 도시에 대한 데이터를 질의하고 싶을 때 상속은 이 문제를 해결할 수 있습니다. cities에서 상속되는 capitals 테이블을 정의합니다.

```
CREATE TABLE cities (
   name     text,
   population    float,
   elevation    int
) ;

CREATE TABLE capitals (
   state     char(2)
) INHERITS (cities);
```

이 경우, capitals 테이블은 그 부모 테이블인 cities 테이블의 열을 모두 상속합니다. 수도는 하나의 추가 열 상태를 가지며 주를 표현합니다.

PostgreSQL에서 1개의 테이블은 0 이상의 테이블로부터 상속하는 것이 가능합니다. 또한 쿼리는 테이블의 모든 행 또는 테이블의 모든 행과 상속된 테이블의 모든 행을 참조할 수 있습니다. 후자가 기본 동작입니다. 예를 들어, 다음 문의는 500 피트보다 높은 고도에 위치한 모든 도시의 이름을 수도를 포함하여 검색합니다.

```
SELECT name, elevation
   FROM cities
   WHERE elevation > 500;
```

이 쿼리는 다음 결과를 출력합니다.

```
name | elevation
-----------+-----------
Las Vegas | 2174
Mariposa | 1953
 Madison | 845
```

다음 쿼리는 500 피트보다 높은 고도에 위치한 모든 도시를 검색합니다.

```
SELECT name, elevation
    FROM ONLY cities
   WHERE elevation > 500;
```

```
name | elevation
-----------+-----------
Las Vegas | 2174
Mariposa | 1953
```

여기서 ONLY 키워드는 쿼리가 cities 테이블만을 대상으로 하고 cities 이하 상속 계층에 있는 테이블은 대상으로 하지 않음을 의미합니다. 지금까지 논의한 많은 명령(SELECT, UPDATE 및 DELETE)이 ONLY 키워드를 지원합니다.

또한 명시적으로 자손 테이블이 포함되어 있음을 나타내기 위해 테이블 이름 뒤에 *을 쓸 수도 있습니다.

```
SELECT name, elevation
    FROM cities*
    WHERE elevation > 500;
```

* 의 기능은 항상 디폴트이기 때문에 기입하지 않아도 됩니다. 그러나 이 구문은 기본값을 변경할 수 있었던 이전 릴리스와의 호환성을 위해 여전히 지원하고 있습니다.

특정 행이 어떤 테이블에서 온 것인지 알고 싶을 수도 있습니다. 각 테이블에는 tableoid라는 원본 테이블을 나타내는 시스템 열이 있습니다.

```
SELECT c.tableoid, c.name, c.elevation
FROM cities
WHERE c.elevation > 500;
```

출력은 다음과 같습니다.

```
tableoid | name | elevation103
-----------+-----------+---------------
139793 | Las Vegas | 2174
 139793 | Mariposa | 1953
 139798 | Madison | 845
```

pg_class와 결합하면 테이블의 실제 이름을 알 수 있습니다.

```
SELECT p.relname, c.name, c.elevation
FROM cities c, pg_class p
```

```
WHERE c.elevation > 500 AND c.tableoid=p.oid;
```

출력은 다음과 같습니다.

```
relname | name | elevation
----------+-----------+-----------
cities | Las Vegas | 2174
cities | Mariposa | 1953
capitals | Madison | 845
```

동일한 효과를 얻는 또 다른 방법은 별칭 regclass를 사용하여 테이블의 OID를 기호적으로 표시하는 것입니다.

```
SELECT c.tableoid::regclass, c.name, c.elevation
FROM cities c
WHERE c.elevation > 500;
```

상위 테이블의 검사 제약 조건과 비 널 제약 조건은 NO INHERIT 절에 명시 적으로 지정되지 않는 한 자식 테이블에 자동으로 상속됩니다. 다른 유형의 제약(유일성 제약, 기본 키, 외래 키 제약)은 상속되지 않습니다.

테이블은 하나 이상의 상위 테이블에서 상속 가능합니다. 이 경우 테이블은 상위 테이블에 정의된 열의 합이 됩니다. 자식 테이블 에서 선언된 열은 이러한 열에 추가됩니다. 부모 테이블에 동일한 이름의 열이 있거나 부모 테이블과 자식 테이블에 같은 이름의 열이 있으면 열이 "통합"되어 하위 테이블에서 단 하나의 열이됩니다. 통합하려면 열은 동일한 데이터 형식을 가져야 합니다. 다른 데이터 유형의 경우 오류가 발생합니다. 상속 가능한 검사 제약과 비 NULL 제약은 유사한 방식으로 통합됩니다. 검사 제약 조건은 이름이 같은 경우 통합되며 조건이 다른 경우 통합에 실패합니다.

테이블 상속은 일반적으로 CREATE TABLE 문의 INHERITS 절을 사용하여 하위 테이블을 만들 때 설정됩니다. 그 밖에도, 호 환성을 가지는 방법으로 정의 끝난 테이블에 새롭게 부모-자식 관계를 붙일 수도 있습니다. 이것은 ALTER TABLE의 INHERIT 형식을 사용합니다. 이를 위해 새 하위 테이블에는 상위 테이블과 이름이 같은 열이 있어야 하며 해당 열의 형식은 동일한 데이터 형식이어야 합니다. 또한 상위 테이블과 동일한 이름과 동일한 표현식에 대한 검사 제한조건이 있어야 합니다. ALTER TABLE의 NO INHERIT 형식을 사용하여 마찬가지로 상속 관계를 하위 테이블에서 제거 할 수 있습니다. 이러한 상속 관계의 동 적 추가, 동적 삭제는 상속 관계를 테이블 분할에 사용하는 경우에 유용합니다.

나중에 자식 테이블로 만들 예정인 호환성을 가진 테이블을 쉽게 만드는 방법 중 하나는 CREATE TABLE에서 LIKE 절을 사용하는 것입니다. 이렇게하면 원래 테이블과 동일한 열을 가진 새 테이블이 만들어집니다. 새 하위 테이블이 항상 상 위 테이블과 일치하는 제약 조건을 가지며 호환 가능한 것으로 간주되도록 원래 테이블에 CHECK 제약 조건이 있으면 LIKE에 INCLUDING CONSTRAINTS 옵션을 지정해야합니다.

하위 테이블이 있으면 부모 테이블을 삭제할 수 없습니다. 또한 하위 테이블에서 상위 테이블에서 상속된 열이나 검사 제약 조건을 삭제하거나 변경할 수 없습니다. 테이블과 모든 하위 테이블을 삭제하려면 CASCADE 옵션을 사용하여 상위 테이블을 삭제하는 것이 쉬운 방법입니다.

ALTER TABLE은 컬럼 데이터 정의 및 점검 제한조건의 변경을 상속 계층 구조의 테이블에 알립니다. 다시, 다른 테이블에 종속된 컬럼의 삭제는 CASCADE 옵션을 사용할 때만 가능합니다. ALTER TABLE은 중복 열의 통합 및 거부에 대해 CREATE TABLE 중에 적용되는 규칙을 따릅니다.

12.2 ALTER TABLE

ALTER TABLE 구문은 테이블을 수정할 수 있는 명령문입니다. 테이블을 만든 후에 실수를 발견하거나 응용 프로그램 요구 사항이 변경된 경우 테이블을 삭제하고 다시 만들 수 있습니다. 그러나 테이블에 데이터가 이미 입력되어 있거나 테이블이 다른 데이터베이스 오브젝트 (예 : 외래 키 제약 조건)에서 참조되는 경우 이는 좋은 방법이 아닙니다. 따라서 기존 테이블을 변경하기 위한 일련의 명령을 제공합니다. 테이블의 데이터를 변경하는 개념이 아닌 테이블의 정의와 구조를 변경하는 데 중점을 둡니다.

12.2.1 Syntax

```
ALTER TABLE 테이블이름

  [ADD 속성이름 데이터타입] -- 컬럼 추가
  [DROP COLUMN 속성이름] -- 컬럼 삭제
  [ALTER COLUMN 속성이름 데이터타입] -- 컬럼 수정
  [ALTER COLUMN 속성이름 데이터타입 [NULL | NOT NULL]] -- 컬럼 수정 (널 허용/비허용)
  [ADD PRIMARY KEY(속성이름)] -- 기본키 추가
  [[ADD | DROP] 제약이름] -- 제약조건 추가 또는 삭제
```

12.2.2 ALTER TABLE ADD

테이블에 컬럼을 추가하는 명령문입니다. 한번의 ADD 명령으로 여러 개의 컬럼 추가 가능하고, 하나의 컬럼만 추가하는 경우에는 괄호를 사용하면 됩니다.추가된 컬럼은 테이블의 마지막 부분에 생성되며 사용자가 컬럼의 위치를 지정할 수 없습니다.그러나 추가된 컬럼에 기본 값은 지정할 수 있습니다. 기존 데이터가 존재하면 추가된 컬럼 값은 NULL로 입력 되고, 새로 입력되는 데이터에 대해서만 기본 값이 적용됩니다.

```
ALTER TABLE '테이블 명'  ADD COLUMN '추가하려는 컬럼 명' '컬럼 데이터 타입';
ALTER TABLE products ADD COLUMN description text;
```

다음 구문을 사용하면 열 제한 조건도 동시에 정의할 수 있습니다.

```
ALTER TABLE products ADD COLUMN description text CHECK(description<>'');

ALTER TABLE products ADD CHECK (name <> '');

ALTER TABLE products ADD CONSTRAINT some_name UNIQUE (product_no);


ALTER TABLE products ADD FOREIGN KEY(product_group_id) REFERENCES product_groups;
```

실제로는 CREATE TABLE내의 열의 기술에 사용되고 있는 모든 옵션을, 여기에서 사용할 수 있습니다. 그러나 기본값은 주어진 제약 조건을 만족해야 합니다. 이를 충족하지 않으면 ADD가 실패합니다. 새 열에 올바르게 값을 넣은 후에 제약 조건을 추가 할 수 있습니다.

NOT NULL 제약 조건은 테이블 제약 조건으로 작성할 수 없으므로 다음 구문을 사용하여 추가합니다.

```
ALTER TABLE products ALTER COLUMN product_no SET NOT NULL;
```

제약 조건이 즉시 검사되므로 제약 조건을 추가하기 전에 테이블의 데이터가 제약 조건에 충족되어야 합니다.

12.2.3 ALTER TABLE ALTER COLUMN

ALTER COLUMN 명령문으로 새로운 기본값을 지정하거나, 해당 컬럼의 데이터 타입 변경 등을 할 수 있습니다. 변경 대상 컬럼에 데이터가 없거나 NULL값만 존재할 경우에 SIZE를 줄일 수 있습니다.

```
ALTER TABLE '테이블 명' ALTER COLUMN '해당 컬럼명' '새로운 데이터 타입';
```

열에 새 기본값을 설정하는 구문은 다음과 같습니다.

```
ALTER TABLE products ALTER COLUMN price SET DEFAULT 7.77
```

이는 테이블의 기존 행에 아무런 영향을 주지 않습니다. 향후 INSERT 명령을 위해 단순히 기본값을 변경하는 것입니다.

기본값을 삭제하려면 다음을 수행합니다.

```
ALTER TABLEproducts ALTER COLUMN price DROP DEFAULT;
```

이것은 기본값을 NULL로 설정하는 것과 같습니다. 따라서 정의되지 않은 기본값을 삭제해도 오류는 발생하지 않습니다. 왜냐하면 NULL 내재적 기본값이기 때문입니다.

열을 다른 데이터 유형으로 변환하는 구문은 다음과 같습니다.

```
ALTER TABLE products ALTER COLUMN price TYPE numeric(10,2);
```

열의 기본값을 새 형식으로 변환하고 동시에 열과 관련된 모든 제약 조건을 새 형식으로 변환하려고 시 도합니다. 그러나 이러한 변환은 실패할 수 있으며 예상을 초과하는 결과를 초래할 수 있습니다. 유형을 변경하기 전에 해당 열에 대한 모든 제약 조건을 제거한 다음 나중에 적절하게 변경된 제약 조건을 재 정의하는 것이 가장 좋습니다.

NOT NULL 제약 조건을 삭제할 경우에는 다음을 수행하십시오.

```
ALTER TABLE products ALTER COLUMN product_no DROP NOT NULL
```

12.2.4 ALTER TABLE RENAME

테이블명을 변경할 수 있습니다.

```
ALTER TABLE '테이블 명' RENAME TO '변경하려는 테이블 명';

ALTER TABLE products RENAME TO items;
```

RENAME명령문으로 특정 컬럼의 이름을 변경할 수 있습니다.

```
ALTER TABLE products RENAME COLUMN product_no TO product_number;
```

12.2.5 ALTER TABLE DROP

열에 있는 모든 데이터를 지웁니다. 또한 해당 열과 관련된 테이블 제한조건도 지워집니다. 그러나 열이 다른 테이블의 외래 키제약 조건으로 참조되는 경우 PostgreSQL은 암시적으로 제약 조건을 지우지 않습니다. CASCADE를 추가하면 열에 의존하는 모든 것을 지울 수 있습니다.

```
ALTER TABLE '테이블 명' DROP COLUMN '삭제할 컬럼명';

ALTER TABLE products DROP COLUMN description CASCADE;
```

제약 조건을 삭제하려면 제약 조건 이름을 알아야 합니다. 스스로 이름을 지정하면 간단합니다. 그러나 자체적으로 이름을 지정하지 않으면 시스템 생성의 이름이 할당되므로 이를 확인해야합니다. 이를 위해 psql의 \d tablename 명령을사용하면편리합니다.

다른 인터페이스도 테이블의 세부사항을 조사하는 방법을 가질 수 있습니다. 제약 이름을 알면 다음 명령으로 제약을 삭제할 수 있습니다.

```
ALTER TABLE products DROP CONSTRAINT some_name;
```

열을 삭제하는 것과 마찬가지로 다른 것이 의존하는 제약 조건을 삭제하려면 CASCADE를 지정해야합니다. 예를 들어, 외래 키 제약 조건은 참조되는 열의 고유 또는 기본 키 제약 조건에 따라 다릅니다.

12.3 DROP TABLE

DROP TABLE은 데이터베이스에서 테이블을 제거합니다. 테이블 소유자, Schema 소유자 및 수퍼유저만 테이블을 삭제할 수 있습니다. 테이블을 삭제하지 않고 행 테이블을 비우려면 DELETE 또는 TRUNCATE 를 사용하십시오 .

DROP TABLE 대상 테이블에 대해 존재하는 모든 인덱스, 규칙, 트리거 및 제약 조건을 항상 제거합니다. 단, 뷰에서 참조하는 테이블이나 다른 테이블의 외래 키 제약 조건을 삭제하려면 을 CASCADE지정해야 합니다. (CASCADE종속 뷰를 완전히 제거하지만 외래 키의 경우 외래 키 제약 조건만 제거하고 다른 테이블은 완전히 제거하지 않습니다.)

12.3.1 Syntax

```
DROP TABLE [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

12.3.2 Example

```
DROP TABLE weather ;
```

12.4 TRUNCATE TABLE

TRUNCATE TABLE은 테이블을 초기화합니다. 용량이 줄어들고 인덱스 등도 모두 삭제됩니다. DELETE보다 수행 속도가 훨씬 빠르나, ROLLBACK을 사용하여 데이터를 복구할 수 없습니다.

12.4.1 Syntax

```
TRUNCATE [ TABLE ] [ ONLY ] name [ * ] [, ... ]
    [ RESTART IDENTITY | CONTINUE IDENTITY ] [ CASCADE | RESTRICT ]
```

12.4.2 Example

```
TRUNCATE TABLE weather ;
```

12.5 주요 오브젝트 DDL

12.5.1 VIEW

view는 사용자에게 접근이 허용된 자료만을 제한적으로 보여주기 위해 하나 이상의 기본 테이블로부터 유도된 이름을 가지는 가상 테이블입니다. 저장장치 내에 물리적으로 존재하지 않지만 사용자에게 실재하는 것처럼 간주됩니다. 주로 데이터 보정작업, 처리과정 시험 등 임시적인 작업을 위한 용도로 활용됩니다.

View는 Join문의 사용 최소화로 사용상의 편의성을 최대화합니다. 필요한 데이터만 뷰로 정의해서 처리할 수 있기 때문에 관리가 용이하고 명령문이 간단해진다는 장점이 있습니다. 논리적 데이터 독립성을 제공하며, 접근 제어를 통한 자동 보안이 제공됩니다.

View를 생성하는 구문은 다음과 같습니다.

```
CREATE VIEW comedy AS SELECT *
   FROM film
   WHERE kind='comedy';
```

film 테이블에 있던 열을 포함하는 View가 생성됩니다.

WITH [CASCADED|LOCAL] CHECK OPTION 옵션은 VIEW의 자동 업데이트를 제어합니다. 이 옵션을 지정하면 새 행이 VIEW 정의 조건을 충족하는지 확인하기 위해 VIEW 명령을 검사합니다.

- LOCAL

  새 행은 VIEW에 직접 정의된 조건에 대해서만 확인됩니다. Base VIEW에 정의된 모든 조건은 확인되지 않습니다.

- CASCADED

  VIEW 및 모든 Base VIEW 조건에 대해 새 행을 확인합니다. 만약 LOCAL 과 CASCADED 옵션 중 선택되지않으면 기본으로 CASCADED 가 적용됩니다.

View를 삭제하는 구문은 다음과 같습니다.

```
DROP VIEW comedy;
```

CASCADE 파라미터를 이용하면 View에 의존하는 객체와 해당 객체의 의존하는 모든 객체를 자동으로 삭제합니다.

```
DROP VIEW comedy CASCADE;
```

12.5.2 INDEX

인덱스는 주로 데이터베이스의 성능을 향상시키는데 사용됩니다. CREATE INDEX는 지정된 relation의 지정된 열에 대한 인덱스를 구성합니다. 인덱스의 키 필드는 열 이름 또는 괄호 안에 작성된 표현식으로 지정됩니다. 인덱스 방법이 다중 열 인덱스를 지원하는 경우 여러 필드를 지정할 수 있습니다. PostgreSQL은 인덱스 메서드 B-tree,GiST, SP-GiST, GIN 및 BRIN을 제공합니다.

WHERE 절이 있으면 부분 인덱스가 생성됩니다. 부분 인덱스는 테이블의 일부, 일반적으로 인덱싱에 더 유용한 부분에 대한 항목만 포함하는 인덱스입니다.

film 테이블의 title열에 고유한 B-tree 인덱스를 생성하는 예는 다음과 같습니다.

```
CREATE UNIQUE INDEX title_idx ON films (title);
```

film 테이블의 title열과 director와 rating 열이 포함된 고유한 B-tree 인덱스를 생성하는 예는 다음과 같습니다.

```
CREATE UNIQUE INDEX title_idx ON films (title) INCLUDE (director, rating);
```

중복 제거가 비활성화된 B-tree 인덱스를 생성하는 예는 다음과 같습니다.

```
CREATE INDEX title_idx ON films (title) WITH (deduplicate_items = off);
```

대소문자를 구분하지 않는 INDEX 검색을 허용하는 표현식에 인덱스를 생성하는 예는 다음과 같습니다.

```
CREATE INDEX ON films ((lower(title)));
```