Sekwencje, IDENTITY, AUTO_INCREMENT

> Klucz sztuczny, opisany na stronie Klucze: główny, obcy, unikalny, naturalny, sztuczny, musi skądś wziąć kolejną wartość — ta strona pokazuje cztery mechanizmy, którymi cztery silniki tej książki generują takie wartości, i dlaczego dziury w numeracji są normalnym efektem ubocznym, nie usterką.

Skróty użyte na tej stronie (1)

Skrót	Co znaczy
SQL	Structured Query Language: język zapytań do relacyjnych baz danych

Fiszka

ZMIENIA SYSTEM

Generator wartości: SERIAL i GENERATED AS IDENTITY (PostgreSQL)

```
CREATE TABLE klienci (
    id INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    nazwa VARCHAR(200) NOT NULL
);
```

`GENERATED ALWAYS AS IDENTITY` to standardowa, zalecana dziś składnia w PostgreSQL — silnik tworzy pod spodem sekwencję powiązaną z kolumną i nadaje z niej kolejną wartość przy każdym `INSERT`, który nie podaje `id` jawnie. Starszy zapis `id SERIAL PRIMARY KEY`, widoczny na stronie Klucze: główny, obcy, unikalny, naturalny, sztuczny, robi dokładnie to samo, tylko starszą składnią sprzed wprowadzenia `IDENTITY` do standardu SQL.

Przykład:

```
CREATE TABLE
```

SQL = Structured Query Language: język zapytań do relacyjnych baz danych · **id** = pokaż użytkownika, jego numer UID oraz grupy, do których należy

ZMIENIA SYSTEM

Generator wartości: SEQUENCE i IDENTITY od 12c (Oracle)

```
CREATE SEQUENCE klienci_id_seq START WITH 1 INCREMENT BY 1;

CREATE TABLE klienci (
    id NUMBER DEFAULT klienci_id_seq.NEXTVAL PRIMARY KEY,
    nazwa VARCHAR2(200) NOT NULL
);
```

Przed wersją 12c Oracle nie miał kolumn generowanych automatycznie w ogóle — trzeba było utworzyć sekwencję osobno i podłączyć ją ręcznie jako wartość domyślną kolumny, jak wyżej. Od wersji 12c dostępna jest też składnia bliższa standardowi:

```
CREATE TABLE klienci (
    id NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    nazwa VARCHAR2(200) NOT NULL
);
```

Przykład:

```
Table created.
```

id = pokaż użytkownika, jego numer UID oraz grupy, do których należy

ZMIENIA SYSTEM

Generator wartości: IDENTITY i SEQUENCE (SQL Server)

```
CREATE TABLE klienci (
    id INT IDENTITY(1,1) PRIMARY KEY,
    nazwa NVARCHAR(200) NOT NULL
);
```

`IDENTITY(1,1)` oznacza start od `1` z krokiem `1` — SQL Server ma też osobny obiekt `SEQUENCE`, niezwiązany bezpośrednio z żadną kolumną, przydatny, gdy ta sama sekwencja wartości ma być współdzielona przez więcej niż jedną tabelę:

```
CREATE SEQUENCE seq_numer_zamowienia START WITH 1000 INCREMENT BY 1;
```

Przykład:

```
Commands completed successfully.
```

SQL = Structured Query Language: język zapytań do relacyjnych baz danych · **id** = pokaż użytkownika, jego numer UID oraz grupy, do których należy

ZMIENIA SYSTEM

Generator wartości: AUTO_INCREMENT (MySQL)

```
CREATE TABLE klienci (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nazwa VARCHAR(200) NOT NULL
);
```

Najprostsza składnia z czterech silników — `AUTO_INCREMENT` przy kolumnie klucza głównego wystarcza, bez osobnego tworzenia sekwencji. MySQL trzyma bieżącą wartość licznika jako właściwość samej tabeli, nie jako niezależny obiekt, który dałoby się współdzielić między tabelami tak jak `SEQUENCE` w pozostałych silnikach.

Przykład:

```
Query OK, 0 rows affected (0.03 sec)
```

id = pokaż użytkownika, jego numer UID oraz grupy, do których należy

Cztery mechanizmy generowania wartości

Silnik	Mechanizm

PostgreSQL	`GENERATED ALWAYS AS IDENTITY` (dawniej `SERIAL`), oparte o `SEQUENCE`
Oracle	`SEQUENCE` + `NEXTVAL` (dowolna wersja), `GENERATED AS IDENTITY` (od 12c)
SQL Server	`IDENTITY(seed, step)` przy kolumnie, albo niezależny obiekt `SEQUENCE`
MySQL / MariaDB	`AUTO_INCREMENT` przy kolumnie, bez osobnego obiektu sekwencji

Odczytanie nadanej wartości

RETURNING (PostgreSQL)

```
INSERT INTO klienci (nazwa) VALUES ('Nowy Klient Sp. z o.o.') RETURNING id;
```

Przykład:

```
 id
----
  7
(1 row)
```

Sposób opisany już przy okazji `INSERT` na stronie INSERT: jeden wiersz, wiele wierszy, INSERT ... SELECT — najbardziej bezpośredni, bo zwraca wartość w tym samym poleceniu, które ją nadało.

currval (PostgreSQL, alternatywa)

```
SELECT currval('klienci_id_seq');
```

`currval` zwraca ostatnią wartość nadaną z danej sekwencji w bieżącej sesji — działa tylko po tym, jak ta sama sesja już raz pobrała wartość z tej sekwencji (`nextval`, wywołane niejawnie przez `INSERT`); wywołane na sekwencji, z której dana sesja jeszcze nic nie pobrała, zwraca błąd.

Przykład:

```
 currval
---------
```

```
       7
(1 row)
```

SCOPE_IDENTITY() kontra @@IDENTITY (SQL Server)

```
INSERT INTO klienci (nazwa) VALUES (N'Nowy Klient Sp. z o.o.');
SELECT SCOPE_IDENTITY() AS nowy_id;
```

Przykład:

```
nowy_id
-----------
7.0


(1 rows affected)
```

`SCOPE_IDENTITY()` zwraca ostatnią wartość `IDENTITY` nadaną w tym samym zakresie wykonania (tej samej procedurze, tym samym poleceniu) — dokładnie tę, którą nadał poprzedzający `INSERT`. `@@IDENTITY` zwraca ostatnią wartość `IDENTITY` nadaną w tej samej sesji, niezależnie od zakresu — jeśli tabela `klienci` ma wyzwalacz, który przy okazji wstawia wiersz do innej tabeli z własnym `IDENTITY`, `@@IDENTITY` zwróci wartość z tamtej, ukrytej operacji, nie z `INSERT`a, który faktycznie nas interesował. Dlatego w tej książce zawsze sięgamy po `SCOPE_IDENTITY()`, nigdy po `@@IDENTITY`.

LAST_INSERT_ID() (MySQL)

```
INSERT INTO klienci (nazwa) VALUES ('Nowy Klient Sp. z o.o.');
SELECT LAST_INSERT_ID();
```

Przykład:

```
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                7 |
+------------------+
1 row in set (0.00 sec)
```

`LAST_INSERT_ID()` zwraca ostatnią wartość `AUTO_INCREMENT` nadaną w bieżącym połączeniu, niezależnie od tego, ile innych sesji wstawia dane równolegle do tej samej tabeli — wartość jest przypisana do połączenia, nie do tabeli globalnie.

Po co to istnieje

> **Pojęcie: klucz sztuczny.** Identyfikator niemający żadnego znaczenia poza samą bazą danych, generowany automatycznie wyłącznie po to, żeby jednoznacznie odróżniać wiersze od siebie — w przeciwieństwie do klucza naturalnego, jak numer NIP, opisanego na stronie Klucze: główny, obcy, unikalny, naturalny, sztuczny. Sekwencje, `IDENTITY` i `AUTO_INCREMENT` to cztery różne, silnikowo specyficzne sposoby generowania kolejnej wartości takiego klucza bez ręcznego liczenia „jaki jest największy `id` w tabeli, dodaj jeden" — podejście, które samo w sobie byłoby wyścigiem przy współbieżności, dokładnie tym samym problemem, który opisuje strona MERGE / UPSERT — jak to robi każdy silnik przy okazji naiwnego sprawdzenia przed wstawieniem.

Jak to działa

Rozwiń: jak to działa pod spodem

Dziury w numeracji i cache sekwencji — dlaczego to normalne. Sekwencja nadaje kolejną wartość w chwili wywołania, niezależnie od tego, czy `INSERT`, który o nią poprosił, w ogóle się powiedzie. `ROLLBACK` transakcji cofa wstawiony wiersz, ale nie cofa pobranej wartości sekwencji — ten numer po prostu przepada, nigdy więcej nie zostanie nadany. To samo dzieje się przy błędzie naruszającym inne ograniczenie już po pobraniu wartości z sekwencji.

```
BEGIN;
INSERT INTO klienci (nazwa) VALUES ('Test');  -- pobiera id = 12
ROLLBACK;
INSERT INTO klienci (nazwa) VALUES ('Kolejny prawdziwy klient');  -- dostaje id = 13, nie
12
```

Dodatkowo wiele silników domyślnie pobiera wartości sekwencji partiami po kilka lub kilkadziesiąt naraz i trzyma je w pamięci sesji (cache sekwencji) dla wydajności — jeśli sesja zakończy się bez wykorzystania całej pobranej partii, te wartości również przepadają. Z obu

powodów kolejne identyfikatory w tabeli produkcyjnej niemal zawsze mają dziury, i to jest zachowanie prawidłowe, nie usterka do naprawienia.

Dlaczego nie wolno traktować id jako licznika. Skoro dziury w numeracji są normalnym efektem ubocznym `ROLLBACK`ów i cache'u, różnica między dwoma kolejnymi wartościami `id` nie mówi nic wiarygodnego o tym, ile wierszy faktycznie powstało między nimi — zapytanie w stylu „`MAX(id) - MIN(id)` to w przybliżeniu liczba klientów" daje wynik zawyżony o nieznaną liczbę nieudanych prób i nigdy nie powinno zastępować zwykłego `COUNT(*)`.

Reset sekwencji po imporcie danych. Import danych z zewnętrznego pliku, który wstawia wiersze z jawnie podanymi wartościami `id` (na przykład przy migracji z innego systemu), nie aktualizuje automatycznie stanu sekwencji powiązanej z tą kolumną — kolejny zwykły `INSERT` bez podanego `id` może spróbować nadać wartość, która już istnieje.

```
SELECT setval('klienci_id_seq', (SELECT MAX(id) FROM klienci));
```

W PostgreSQL `setval` ustawia sekwencję na wartość co najmniej równą największemu istniejącemu `id`, tak żeby kolejne wywołanie `nextval` nie kolidowało z zaimportowanymi danymi. Oracle wymaga w tym celu usunięcia i ponownego utworzenia sekwencji z nowym `START WITH`, SQL Server używa `DBCC CHECKIDENT`, MySQL — `ALTER TABLE klienci AUTO_INCREMENT = wartosc`.

Częste błędy

Poleganie na @@IDENTITY zamiast SCOPE_IDENTITY() w SQL Server.

Wyzwalacz na tabeli `klienci`, który przy okazji wstawia wiersz audytowy do innej tabeli z własną kolumną `IDENTITY`, sprawia, że `@@IDENTITY` po `INSERT`cie do `klienci` zwraca wartość z tabeli audytowej, nie z `klienci` — błąd trudny do wykrycia, bo pojawia się tylko wtedy, gdy taki wyzwalacz akurat istnieje, i znika bez śladu, gdy się go usunie.

Traktowanie różnicy id jako liczby wstawionych wierszy.

```
SELECT MAX(id) - MIN(id) + 1 AS przyblizona_liczba_klientow FROM klienci;
```

Wynik tego zapytania zwykle jest większy niż prawdziwy `COUNT(*)`, bo nie uwzględnia dziur po `ROLLBACK`ach i niewykorzystanym cache sekwencji — jedynym poprawnym sposobem policzenia wierszy jest `COUNT(*)`.

Import danych z jawnym id bez odświeżenia sekwencji.

```
ERROR:  duplicate key value violates unique constraint "klienci_pkey"
DETAIL:  Key (id)=(5) already exists.
```

Klasyczny efekt zaimportowania danych z jawnymi wartościami `id`, po czym pierwszy zwykły `INSERT` bez podanego `id` próbuje nadać wartość, którą sekwencja „pamięta" jako jeszcze niewykorzystaną, mimo że import już ją zajął. Naprawa to reset sekwencji opisany w sekcji „Jak to działa".

□□ Ćwiczenia

> Zadania sprawdzające.

Ćwiczenie 1 — dziury w numeracji

Tabela `klienci` ma sekwencję, z której nadano dotąd dwadzieścia wartości, ale `SELECT COUNT(*) FROM klienci` zwraca `17`. Czy to oznacza uszkodzenie danych, i jakie dwie przyczyny mogły spowodować tę różnicę bez żadnego błędu w bazie?

Odpowiedź

Nie oznacza uszkodzenia danych. Dwie najbardziej prawdopodobne przyczyny to transakcje zakończone `ROLLBACK`iem po pobraniu wartości z sekwencji (wartość przepadła, wiersz nigdy nie powstał) oraz cache sekwencji, z którego część pobranych wartości nie została wykorzystana, zanim sesja się zakończyła. Obie sytuacje są normalnym efektem ubocznym działania sekwencji, opisanym w sekcji „Jak to działa" tej strony.

Pytania kontrolne

1. Dlaczego `@@IDENTITY` w SQL Server jest ryzykowniejszy od `SCOPE_IDENTITY()`?

Odpowiedź

`@@IDENTITY` zwraca ostatnią wartość `IDENTITY` nadaną w całej sesji, niezależnie od tego, które polecenie ją nadało — jeśli między naszym `INSERT`em a odczytem wykona się jakikolwiek wyzwalacz wstawiający wiersz do innej tabeli z własnym `IDENTITY`, `@@IDENTITY` zwróci tamtą wartość zamiast naszej. `SCOPE_IDENTITY()` ogranicza się do tego samego zakresu wykonania, więc zawsze zwraca wartość z polecenia, które faktycznie nas interesuje.

2. Dlaczego import danych z jawnie podanymi wartościami klucza głównego wymaga potem ręcznego zresetowania sekwencji?

Odpowiedź

Sekwencja nadaje kolejne wartości niezależnie od tego, czy dana wartość klucza głównego została już użyta przez inny mechanizm — `INSERT` z jawnie podanym `id` w ogóle nie pyta sekwencji o nic, więc sekwencja nie wie, że ta wartość jest już zajęta. Bez ręcznego przesunięcia sekwencji do wartości co najmniej równej największemu zaimportowanemu `id`, kolejny zwykły `INSERT` może spróbować nadać wartość, która już istnieje w tabeli.

Powiązane: Klucze: główny, obcy, unikalny, naturalny, sztuczny · INSERT: jeden wiersz, wiele wierszy, INSERT ... SELECT · MERGE / UPSERT — jak to robi każdy silnik · BEGIN, COMMIT, ROLLBACK, SAVEPOINT

Źródła: dokumentacja PostgreSQL, rozdział „Sequence Manipulation Functions” i „Identity Columns” · dokumentacja Oracle, „CREATE SEQUENCE” i „Identity Columns” · dokumentacja Microsoft, „IDENTITY (Property)” i „SCOPE_IDENTITY (Transact-SQL)” · dokumentacja MySQL, „AUTO_INCREMENT Handling in InnoDB”

Revision #46
Created 2026-09-24 22:42:20 UTC by Tomasz Konieczny
Updated 2026-10-26 14:11:23 UTC by Tomasz Konieczny