JURNAL DASI ISSN: 1411-3201
Vol. 11 No. 2 Juni 2010

IMPLEMENTASI CONSTRAINT UNTUK MENJAMIN KONSISTENSI DAN INTEGRITAS DATA DALAM DATABASE

M. Rudyanto Arief
Dosen STMIK AMIKOM Yogyakarta

Abstract

All of the popular RDBMS products provide table check constraints: Oracle, Informix, DB2, Sybase, and Microsoft SQL Server. Constraints enable enhanced data integrity without requiring procedural logic (such as in stored procedures and triggers).

Keywords: *Primary key, foreign key, check, not null, unique constraint, integritas data, konsistensi data.*

Pendahuluan

Konstraint/ batasan dalam database pada dasarnya adalah pembatasan nilai-nilai yang diperbolehkan untuk diisikan dalam sebuah kolom atau kolom-kolom yang terdapat dalam sebuah tabel. Dengan menerapkan batasan-batasan dalam sebuah tabel maka dapat dipastikan bahwa hanya data-data yang sesuai dengan batasan tersebut saja yang dapat di sikan dalam tabel tersebut. Fungsi utama dari penerapan batasan-batasan/ constraint dalam sebuah tabel adalah untuk menjamin konsistensi dan integritas data-data yang diisikan dalam sebuah tabel. Beberapa waktu yang lalu sebelum basis data berkembang seperti saat ini, banyak programmer dan Database Administrator (DBA) menerapkan validasi data pada sisi client/ program aplikasi dan tidak pada sisi pemrograman servernya.

Tentunya hal ini sangat memerlukan keahlian pemrograman client yang cukup baik dari seorang programmer jika ingin menerapkan validasi yang cukup ketat untuk setiap data-data yang nantinya akan diisikan oleh seorang user dari sisi antarmuka aplikasi clientnya. Contohnya adalah dengan menggunakan visual basic seorang programmer akan membuat listing program tambahan untuk

melakukan pengecekan apakah seorang user sudah mengisikan datanya sesuai dengan yang seharusnya diatur dalam business rules-nya. Seiring dengan berkembangnya database, maka saat ini banyak vendor/ perusahaan pembuat aplikasi database seperti Oracle, Microsoft mulai membuat Database Management System (DBMS) Server yang didalamnya terdapat fitur-fitur yang dapat melakukan validasi terhadap data-data yang dapat diisikan pada sisi server dan bukan lagi pada sisi aplikasi client seperti yang selama ini digunakan oleh programmer. Fitur tersebut disebut dengan constraint. Berikut adalah beberapa contoh constraint yang terdapat dalam produk DBMS, yaitu: PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, NOT NULL. Tentunya jenis-jenis constraint tersebut dapat berbeda untuk masing-masing DBMS yang ada di pasaran saat ini. Constraint dapat di buat pada level tabel atau level kolom, kecuali untuk constraint NOT NULL yang hanya dapat dibuat pada level kolom.

Pembahasan

Primary Key

PRIMARY KEY atau Kunci Utama dalam sebuah tabel merupakan kunci yang akan membatasi pengisian record dalam sebuah tabel agar tidak duplikat (redundant). Syarat sebuah kolom/ field dijadikan PRIMARY KEY dalam sebuah tabel adalah unik dan tidak boleh kosong (NOT NULL). Artinya bahwa nilai record-record dalam kolom yang dijadikan PRIMARY KEY haruslah unik satu sama lain dan nilainya tidak boleh di kosongkan. Jika sebuah kolom di beri constraint PRIMARY KEY maka sudah pasti kolom tersebut nilainya tidak boleh kosong atau harus berisi data.

Berikut adalah perintah dasar pembuatan constraint PRIMARY KEY:

```
CONSTRAINT pk_(nama-tabel)_(nama-kolom) [jenis_constraint]
```

Keterangan:

PK = Singkatan dari jenis constraint yaitu PRIMARY KEY

Nama-tabel = Nama tabel tempat constraint tersebut di buat

Nama-kolom = Nama kolom yang akan di beri constraint
Jenis_constraint = Jenis-jenis constraint yang akan dibuat (PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL, CHECK.

Berikut adalah contoh perintah pembuatan constraint **PRIMARY KEY** pada kolom **nip** dengan nama constraint-nya **pk_dosen_nip** di tabel **dosen** pada level kolom.

```
CREATE TABLE dosen (
nip INTEGER
CONSTRAINT pk_dosen_nip PRIMARY KEY,
nama_dosen VARCHAR(45),
alamat_dosen VARCHAR(255)
);
```

Berikut contoh pembuatan constraint **PRIMARY KEY** pada level tabel untuk kolom **nip** di tabel **dosen**.

```
CREATE TABLE dosen (
nip INTEGER,
nama_dosen VARCHAR(45),
alamat_dosen VARCHAR(255)
CONSTRAINT pk_dosen_nip PRIMARY KEY
);
```

Foreign Key

FOREIGN KEY atau KUNCI TAMU merupakan kunci yang digunakan sebagai penghubung antara satu tabel dengan tabel lainnya. Jika sebuah tabel memiliki FOREIGN KEY maka tabel tersebut dapat di sebut juga dengan tabel kedua dimana tabel utamanya adalah tabel yang memiliki kolom PRIMARY KEY dari FOREIGN KEY tersebut. Perintah dasarnya:

```
CONSTRAINT fk_(nama-tabel)_(nama-kolom) FOREIGN KEY
```

```
REFERENCES tabel-referensi(kolom-referensi)
```

Keterangan:
FK = Singkatan dari FOREIGN KEY
Nama-Tabel = Nama tabel tempat FOREIGN KEY tersebut dibuat.
Nama-Kolom = Nama Kolom yang akan dijadikan FOREIGN KEY
FOREIGN KEY = Jenis constraint-nya, yaitu FOREIGN KEY
REFERENCES = Kata kunci untuk merelasikan tabel ini ke tabel utamanya (tabel yang memiliki PRIMAR KEY)
Tabel-Referensi = Tabel yang akan dijadikan referensi (tabel utama).
Kolom-Referensi = Kolom yang menjadi referensi dari tabel utama.

Berikut adalah contoh penggunaan FOREIGN KEY pada level kolom dan pada level tabel:

```
CREATE TABLE mahasiswa
(
nim INTEGER CONSTRAINT pk_mahasiswa_nim PRIMARY KEY,
nip INTEGER
CONSTRAINT fk_mahasiswa_nim
FOREIGN KEY REFERENCES dosen(nip),
nama_mhs VARCHAR(45),
alamat_mhs VARCHAR(255)
);
```

```
CREATE TABLE mahasiswa
(
nim INTEGER CONSTRAINT pk_mahasiswa_nim PRIMARY KEY,
nip INTEGER
nama_mhs VARCHAR(45),
alamat_mhs VARCHAR(255),
CONSTRAINT fk_mahasiswa_nim
```

```
FOREIGN KEY REFERENCES dosen(nip)
);
```

Pada constraint FOREIGN KEY ada beberapa fitur tambahan yang dapat d berikan pada sebuah kolom dalam tabel, yaitu:

[ON DELETE {CASCADE | NO ACTION}]

Pilihan ini akan menentukan aksi apa yang akan di lakukan terhadap sebuah baris/ record dalam tabel, jika record tersebut memiliki hubungan referensi (referential relationship) dan record yang di referensi di hapus (DELETE) dari tabel induk. Defaultnya adalah NO ACTION.

Jika pilihan CASCADE di gunakan, sebuah record akan di hapus dari tabel referensinya jika record tersebut di hapus dari tabel induknya. Jika pilihan NO ACTION yang digunakan, SQL Server akan menampilkan pesan kesalahan dan proses penghapusan record pada tabel induk di batalkan (ROLLED BACK).

Contohnya, pada database **akademik_xxxx**, tabel **mahasiswa** memiliki hubungan referensi (referential relationship) dengan tabel **dosen**. **FOREIGN KEY mahasiswa.nik** memiliki hubungan referensi dengan **PRIMARY dosen.nik**.

Jika perintah **DELETE** di lakukan pada sebuah record di tabel **dosen**, dan properties **ON DELETE CASCADE** di berikan pada kolom **mahasiswa.nik**, SQL Server melakukan pengecekan terhadap satu atau lebih record yang memiliki ketergantungan di tabel **mahasiswa**. Jika ada record yang memiliki ketergantungan tersebut, maka record tersebut akan di hapus dari tabel **mahasiswa** sesuai dengan record yang di hapus di tabel **dosen**.

Dengan kata lain, jika pilihan **NO ACTION** yang digunakan, SQL Server akan menampilkan pesan kesalahan dan membatalkan penghapusan data untuk record di tabel **dosen** tersebut jika terdapat paling tidak satu record di tabel **mahasiswa** yang berhubungan dengan record tersebut.

[ON UPDATE {CASCADE | NO ACTION}]
Sama dengan [ON DELETE {CASCADE | NO ACTION}], yang membedakan jika **ON DELETE** aksi dilakukan jika record di hapus (DELETE), pada **ON UPDATE** aksi dilakukan jika record di rubah (UPDATE).

Berikut adalah contoh pemberian pilihan ON DELETE CASCADE dan ON UPDATE CASCADE pada sebuah tabel untuk kolom yang menjadi FOREIGN KEY.

```
CREATE TABLE mahasiswa
(
nim INTEGER CONSTRAINT pk_mahasiswa_nim PRIMARY KEY,
nip INTEGER
CONSTRAINT fk_mahasiswa_nim
FOREIGN KEY REFERENCES dosen(nip)
ON DELETE CASCADE ON UPDATE CASCADE,
nama_mhs VARCHAR(45),
alamat_mhs VARCHAR(255)
);
```

Unique

Constraint UNIQUE merupakan sebuah constraint yang akan membatasi pengisian record yang sama kedalam sebuah kolom jika kolom tersebut di berikan constraint UNIQUE dalam sebuah tabel. Constraint ini hampir sama dengan PRIMARY KEY, yaitu menjamin bahwa setiap nilai record yang ada dalam sebuah kolom UNIQUE tidak boleh ada yang sama (unik satu sama lain).

UNIQUE dapat dibuat pada level kolom ataupun level tabel. Sama dengan constraint PRIMARY KEY, FOREIGN KEY, CHECK. Perintah dasarnya:

```
CONSTRAINT UQ_(nama-tabel)_(nama-kolom) UNIQUE
```

Keterangan:
UQ = Singkatan dari UNIQUE
Nama-Tabel = Nama tabel tempat UNIQUE tersebut dibuat.
Nama-Kolom = Nama Kolom yang akan dijadikan UNIQUE
UNIQUE = Jenis constraint-nya, yaitu UNIQUE
Berikut adalah contoh penggunaan UNIQUE dan PRIMARY KEY secara bersamaan pada level kolom:

```
CREATE TABLE dosen (
nip INTEGER
CONSTRAINT pk_dosen_nip PRIMARY KEY,
nama_dosen VARCHAR(45) CONSTRAINT UQ_dosen_nama
UNIQUE,
alamat_dosen VARCHAR(255)
);
```

Not Null

NOT NULL merupakan constraint yang digunakan untuk menjamin pengisian record ke sebuah tabel agar nilai record tersebut harus berisi data. Sebuah kolom jika di berikan constraint NOT NULL, maka kolom tersebut harus berisikan nilai untuk recordnya dan tidak boleh di kosongkan. NOT NULL hanya bisa di definisikan dalam sebuah tabel pada level kolom dan tidak bisa pada level tabel. Perbedaan antara NOT NULL dan PRIMARY KEY adalah bahwa setiap kolom yang dijadikan PRIMARY KEY pasti NOT NULL dan tidak berlaku sebaliknya.

Berikut adalah contoh penggunaan constraint NOT NULL:

```
CREATE TABLE dosen (
nip INTEGER
CONSTRAINT PK_dosen_nip PRIMARY KEY NOT NULL,
nama_dosen VARCHAR(45),
```

```
alamat_dosen VARCHAR(255)
);
```

Check

CHECK merupakan constraint yang memaksa integritas domain dengan membatasi pemasukan nilai yang mungkin di masukkan ke dalam sebuah kolom atau beberapa kolom. Hampir sama dengan constraint FOREIGN KEY dalam hal bagaimana mengendalikan nilai yang akan di masukkan kedalam sebuah kolom. Perbedaannya adalah pada bagaimana keduanya menentukan nilai mana yang valid. FOREIGN KEY mendapatkan daftar nilai valid dari tabel yang lain, sementara CHECK menentukan nilai valid dari ekspresi logika yang tidak menggunakan data yang berdasarkan data pada kolom yang lain. Contohnya, sangat mungkin bagi kita membatasi jangkauan nilai untuk kolom gaji dengan membuat constraint CHECK yang membolehkan data yang memiliki jangkauan dari $15,000 hingga $100,000. Pembatasan ini akan mencegah pengisian gaji di luar jangkauan nilai gaji normal. Untuk kasus tersebut dapat dibuat constraint CHECK dengan menggunakan ekspresi logika (BOOLEAN) yang mengembalikan nilai TRUE atau FALSE berdasarkan operator logika yang dibuat. Berikut adalah contoh ekspresi logikanya:

```
Gaji >= 15000 AND salary <= 100000
```

Dalam satu kolom dapat di terapkan lebih dari satu constraint CHECK. Pengecekannya dilakukan berdasarkan urutan pembuatan constraint CHECK tersebut. Selain itu satu constraint CHECK dapat di terapkan di lebih dari satu kolom (multiple columns) dengan cara membuatnya pada level tabel. Contohnya, Constraint Multiple CHECK dapat digunakan untuk memastikan bahwa tiap record dalam kolom **country** dengan nilai **USA** juga memiliki 2 karakter nilai pada kolom **state**. Hal ini memungkinkan pengecekan kondisi lebih dari satu dalam satu tempat.

Berikut adalah contoh penggunaan constraint CHECK dalam sebuah tabel:

```
CREATE TABLE rekening (
No_rekening CHAR(15) CONSTRAINT pk_rekening_no_rekening NOT NULL,
Kode_cabang CHAR(5)
CONSTRAINT fk_rekening_kode_cabang
FOREIGN KEY REFERENCES cabang_bank(kode_cabang),
Pin CHAR(6),
Saldo MONEY CONSTRAINT chk_rekening_saldo CHECK (saldo > 50000)
```

Penutup

Constraint dapat digunakan pada semua RDBMS yang ada di dunia saat ini seperti ORACLE, MSQL SERVER, PostGreSQL, MySQL. Dengan menerapkan constraint dalam tabel-tabel yang ada dalam sebuah database maka data-data yang diisikan dalam tabel tersebut dapat dijamin konsistensi dan integritasnya hal ini disebabkan karena hanya data-data yang sesuai dengan constraint sajalah yang diijinkan untuk masuk kedalam tabel tersebut. Tentunya penerapan constraint yang tepat untuk masing-masing tabel sangat tergantung dari hasil analisis dan perancangan tabel yang sesuai dengan aturan main/ business rules dari sebuah sistem. Penerapan constraint pada sisi database server tentu saja dapat mempercepat performa dari sebuah sistem jika sistem tersebut memiliki arsitektur client-server. Karena semua proses validasi data tidak dibebankan lagi pada sisi client seperti yang selama ini dilakukan oleh banyak programmer dengan menambahkan algoritma validasi data pada listing program aplikasi client mereka. Selain itu penerapan constraint pada sisi server tentu saja memudahkan seorang DBA dalam memelihara sistem aplikasi. Jika terjadi perubahan aturan main yang berhubungan dengan validasi data maka seorang DBA cukup merubah/ update constraint yang ada

di sisi server (database) dan tidak perlu melakukan perubahan algoritma pada listing program yang ada pada program aplikasi client.

Daftar Pustaka

Arief Rudyanto, Pemrograman Basis Data dengan Transact-SQL menggunakan SQL Server, Penerbit Andi Yogyakarta, 2005.