

Advanced Constraints SQL

by

Joe Celko

copyright 2007

Abstract

- The talk is a short overview of the options a programmer to use DDL (Data Declaration Language) in SQL to enforce a wide range of business rules

Biography

- **Joe Celko**
Author and Consultant
- Joe Celko is an independent consultant and author living in Austin, Texas.
- He is the author of six books on SQL for Morgan-Kaufmann and was a member of the ANSI X3H2 Database Standards Committee from 1987 to 1997. He has written over 750 columns in the computer trade and academic press, mostly dealing with data and databases.

DDL is underused

- There are a lot of things that can be done with DDL to enforce business rules ...
- But people don't do it
 - Programmers still think a column is a field and do not understand the power they have
 - File layouts are easier to write in a hurry than good DDL
- Remember that a smart optimizer will put all these constraints into the execution plan!
 - Constraints become predicates
 - TRIGGERS and STORED PROCEDUREs tell the optimizer nothing

CREATE TABLE Statement

- **CREATE TABLE <name>**
(<columns> [<constraints>])
- **Schema is created all at once**
 - **This can mean circular references**
- **Table names are unique in a schema**
- **DBA can CREATE, DROP or ALTER tables**
 - **Users cannot change database structures**
- **Users INSERT, UPDATE and DELETE data in the tables**
 - **The constraints restrict what they can do at the database level**
 - **Files have no restrictions**

Column Definitions

- **CREATE TABLE <name>**
(<column list> [<constraints>], ...);
- **<Column name> <datatype> [constraints]**
- **Column name is unique in table**
 - **But *should* be used in other table**
 - **Data elements should have one and only one name in the schema**
- **Constraints reference columns in the table where they are declared in most products**
 - **In full SQL-92, they can also reference other tables or views**

Column Constraints

- **NOT NULL** - column only
- **DEFAULT <constant>** - column only
- **CHECK (<search condition>)** – column, table or schema level
- **UNIQUE and PRIMARY KEY** – table level
- **REFERENCES Clause** - schema level
- **Multi-column forms of some of these constraints**

NOT NULL Constraint

- Says this column cannot have a NULL in it
 - Use this constraint automatically, then change it when you have a good reason
 - Historical reasons this is not automatic
- Try not to use it
 - Look for a natural default value in the domain of the column
- If a column is NULL-able, then explain what a NULL means in that column
 - If it can have more than one meaning, you are in trouble
 - You need to know *WHY* the missing value is missing!

DEFAULT Clause

- **DEFAULT <constant or system call>**
 - System calls are **CURRENT_TIMESTAMP**, **CURRENT_USER**, etc
- **Specifies a value for INSERT INTO statements when no value is given**
 - The “default DEFAULT” is **NULL**, unless declared as **NOT NULL**
- **Can be used in UPDATE statements explicitly**
- **Make sure the datatypes match without a forced conversion**
 - **INTEGER NOT NULL DEFAULT 0**
 - **INTEGER NOT NULL DEFAULT 0.00 – conversion!**
 - **CHAR(n) and VARCHAR(n)**

CHECK () Constraint -1

- CHECK (<search condition >)
- References any column in the same row
- All rows in the table must test TRUE or UNKNOWN for the Boolean expression
 - big difference from the WHERE clause!
- Full SQL-92 allows you to reference other tables in schema, but this is not yet common
- Full SQL-92 also has CREATE ASSERTION statements
 - A CHECK() constraint outside the tables, at the database schema level
 - All constraints are true inside an empty table
- This is the place for business rules

CHECK () Constraint -2

- Any predicate can go into a CHECK() constraint
 - CHECK (sex IN (0,1,2,9))
 - CHECK (birthdate < CURRENT_TIMESTAMP)
 - CHECK (IQ >= 0)
 - CHECK (birthdate < hire_date)
 - CHECK (partcode LIKE 'P-%')
 - CHECK (x BETWEEN 0 and 42)
- Remember that a smart optimizer will put all these constraints into the execution plan
- TRIGGERS and STORED PROCEDURES tell the optimizer nothing

CHECK () Constraint -3

- CHECK() constraints can be given names which will appear in error messages.
- The syntax is simply "CONSTRAINT <name> CHECK (<search condition>)"
 - The <name> is global, not local to the table.
- You can write a monster predicate in one CHECK() clause, but then you have to figure out which part of the monster predicate is causing the error.
- Better to have several short, well named constraints

CHECK () Constraint -4

- CHECK() constraints can use CASE expressions for complex logic

```
CHECK (CASE WHEN x =1 THEN 'T'  
          WHEN x =2 AND y=1  
            THEN 'F'  
          ...  
          ELSE 'F' END = 'T')
```

- This is where you put business rules in a table.

CHECK () Constraint -5

- While not widely available, you can reference the table itself in subquery predicates
 - CHECK ((SELECT MAX(seq) FROM FooBar)
= (SELECT COUNT(*) FROM FooBar))
 - CHECK (NOT EXISTS
(SELECT * FROM Personnel
GROUP By dept
HAVING COUNT(*) >= 100))
- All constraints are TRUE for an empty table

Constant Tables -1

- Trick: table to store physical constants, or other read-only data
- **CREATE TABLE PhysicalConstants**
(keycol CHAR(1) NOT NULL PRIMARY KEY
DEFAULT 'X'
CHECK(keycol = 'X'),
pi FLOAT NOT NULL DEFAULT 3.141592653,
e FLOAT NOT NULL DEFAULT 2.718282, ...);
- **INSERT INTO PhysicalConstants DEFAULTS** makes the defaults appear
- **CHECK()** allows only one row

Constant Tables -2

- SQL-92 Version !
- **CREATE VIEW PhysicalConstants (pi, e, ..)
VALUES (CAST (3.141592653 AS FLOAT),
 CAST(2.718282 AS FLOAT),
 ...);**
- The CAST() forces the data type

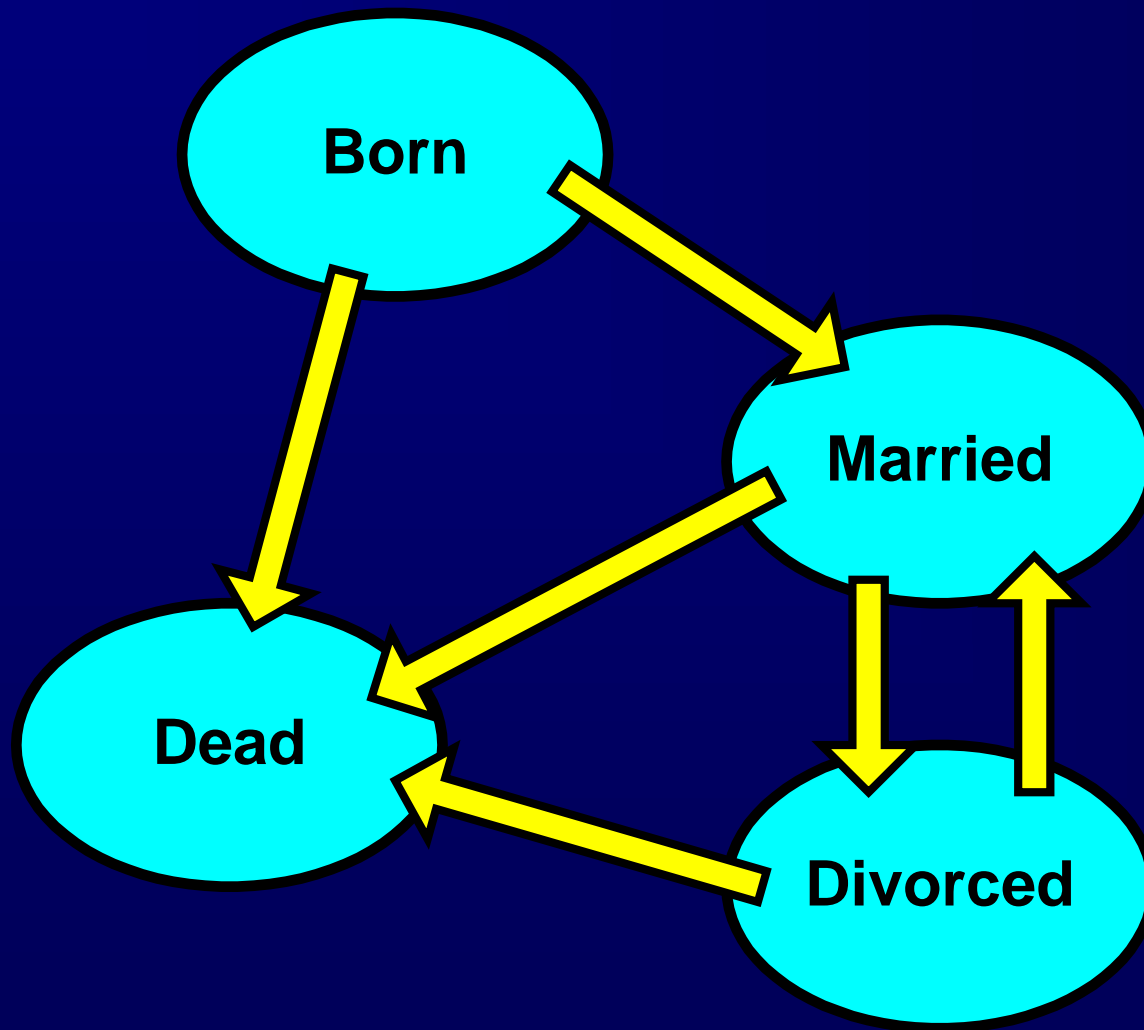
Transition Constraints -1

- Transition constraints can be done with an Auxiliary table
 - Can include duration between state changes and other data

```
CREATE TABLE Transitions
(prior_state INTEGER NOT NULL,
 current_state INTEGER NOT NULL,
PRIMARY KEY (prior_state, current_state),
etc.);
```

```
CREATE TABLE Process
(..,
 FOREIGN KEY (prior_state, current_state)
 REFERENCES Transitions (prior_state, current_state),
..);
```

Transition Constraints -2



Transition Constraints -3

- (Born, Born) -- initial state
- (Born, Dead)
- (Born, Married)
- (Married, Divorced)
- (Married, Dead)
- (Divorced, Married)
- (Divorced, Dead)
- (Dead, Dead) -- terminal state

UNIQUE Constraints -1

- Two forms: PRIMARY KEY() and UNIQUE()
- Zero or one PRIMARY KEY per table
 - No key means it is not really a table
- PRIMARY KEY(<column list>) is automatically both UNIQUE and NOT NULL
- UNIQUE (<column list>) can have NULLs in the column
 - Not usually a good idea.
- Both are used by the REFERENCES constraint (later)
- A table can have more than one UNIQUE constraint and they can overlap each other

UNIQUE Constraints -2

```
CREATE TABLE Marriage  
(husband CHAR(15) NOT NULL UNIQUE,  
wife CHAR(15) NOT NULL UNIQUE);
```

```
CREATE TABLE Polygamy  
(husband CHAR(15) NOT NULL,  
wife CHAR(15) NOT NULL UNIQUE);
```

UNIQUE Constraints -3

```
CREATE TABLE Polyandry  
(husband CHAR(15) NOT NULL,  
wife CHAR(15) NOT NULL UNIQUE);
```

```
CREATE TABLE CorporateMarriages  
(husband CHAR(15) NOT NULL,  
wife CHAR(15) NOT NULL,  
PRIMARY KEY(husband, wife));
```

UNIQUE Constraints -4

- Example:teacher's schedule
- CREATE TABLE Schedule
(teacher VARCHAR(15) NOT NULL,
class CHAR(15) NOT NULL,
room INTEGER NOT NULL,
period INTEGER NOT NULL,
PRIMARY KEY (teacher, class, room, period));
- That primary key is the most obvious one -- use all the columns

UNIQUE Constraints -5

- The rules we want to enforce are:
- 1) A teacher is in only one room each period
- 2) A teacher teaches only one class each period
- 3) A room has only one class each period
- 4) A room has only one teacher in it each period

UNIQUE Constraints -7

- **CREATE TABLE Schedule_1 -- *version one, wrong!***
(...UNIQUE (teacher, room, period), -- rule #1
UNIQUE (teacher, class, period), -- rule #2
UNIQUE (class, room, period), -- rule #3
UNIQUE (teacher, room, period), -- rule #4
PRIMARY KEY (teacher, class, room, period));
- **Insert three rows and give me a bad 6th period**
('Celko', 'Database 101', 222, 6)
('Celko', 'Database 102', 223, 6)
('Ms Shields', 'Database 101', 223, 6)

UNIQUE Constraints -8

- **CREATE TABLE Schedule_2 -- corrected version**
(...
UNIQUE (teacher, period), -- rules #1 and #2
UNIQUE (room, period)); -- rules #3 and #4
- **If a teacher is in only one room each period, then given a period and a teacher I should be able to determine only one room**
- **If a teacher teaches only one class each period, then class is dependent upon the combination of teacher and period**
- **The same logic holds for the last two rules: class is dependent upon the combination of room and period, and teacher is dependent upon the combination of room and period**

REFERENCES Clause -1

- Single or multi-column syntax
 - **<column declaration> REFERENCES <table 2> [(column list)],**
 - **FOREIGN KEY <column list> REFERENCES <table 2> [(column list)]**
- The column(s) in this table has matching value(s) in <table 2>
- The default referenced column(s) list is the PRIMARY KEY in <table 2>
- You can reference any UNIQUE constraint's column list

REFERENCES Clause -2

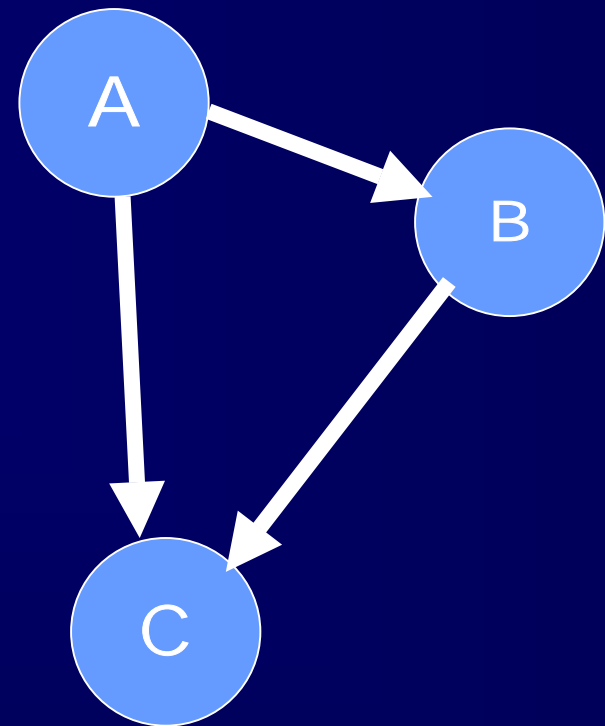
- REFERENCES clause can also have actions
- ON DELETE [NO ACTION | CASCADE | SET NULL | SET DEFAULT]
- ON UPDATE [NO ACTION | CASCADE | SET NULL | SET DEFAULT]
- When the referenced table is changed, this option will take action in the referencing table

REFERENCES Clause -3

- You can cascade all over the schema, which can be very handy
 - a lot of work that would have been in application code is now moved to the database
 - Business rules are in one place
- You can cascade all over the schema, which can be very dangerous
 - It can take a lot of time to execute and lock the database
 - If you have circular references, you can loop forever
 - If you did not know what you were doing, it's too late now

REFERENCES Clause -4

- Good rule is to avoid any circular references
- This also means self-references
 - Change to A causes change to B and to C
 - Change to B causes change to C
 - Does C now reflect the actions of A or B?



CREATE DOMAIN Statement

- **CREATE DOMAIN <domain name>
AS <datatype> <constraints>**
- **Part of SQL-92, but not widely implemented**
 - **Might see vendor version of “user defined data types” (UDT)**
- **It is a “column declaration Macro”**
 - **It does not define new operators on the domain**
 - **You use it in CREATE TABLE statements like a datatype on a column declaration**
- **The ALTER DOMAIN statement allows you to change all the occurrences in the entire schema**
 - **Example: ten digit UPC codes are going to be replaced by a 15 digit GTIN code**

Classes in SQL

```
CREATE TABLE Class
(item_key INTEGER NOT NULL PRIMARY KEY
 sub_class CHAR(1) NOT NULL
      CHECK(sub_class IN ('A', 'B', 'C'),
 ... );
```

```
CREATE TABLE SubClassA
(item_key INTEGER NOT NULL,
 sub_class CHAR(1) NOT NULL CHECK(sub_class = 'A'),
 PRIMARY KEY (item_key, sub_class),
 FOREIGN KEY (item_key, sub_class)
      REFERENCES Class (item_key, sub_class),
 ... );
```

You can also use an optional `UNIQUE()` on `item_key` to get an index

Using VIEWS for Table Constraints

- Technically, a CHECK() can reference any table in the schema with any predicate

```
CREATE TABLE Foobar
(..
CONSTRAINT only_ten_members
CHECK ((SELECT COUNT(*) FROM Foobar) <= 10),
..);
```

- Most products do not have this feature yet.
Fake it with a VIEW..WITH CHECK OPTION

```
CREATE VIEW NewFoobar
AS
SELECT * FROM Foobar
WHERE (SELECT COUNT(*) FROM Foobar) <= 10
WITH CHECK OPTION;
```

Summary

- Programmers still think in procedural code
- CHECK() constraints are good
- You can use tables and DRI actions to build declarative constraints
- Most bad DML (queries) are attempts to repair bad DDL

Questions & Answers

?