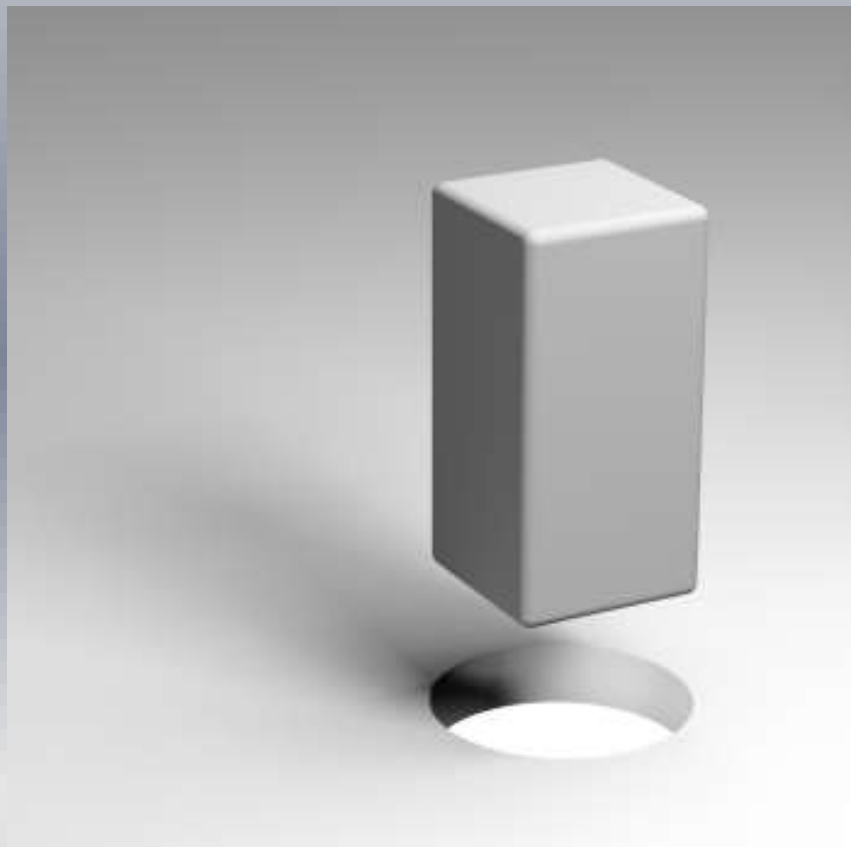
SQL Antipatterns

Krishnakumar S



Design Patterns

- A **solution** to a recurring **problem**, in a **context**
- Abstracts a recurring design structure
- Distills design experience
- Pattern consists of a problem and a solution

Antipatterns

- A **solution** to a recurring **problem**, in a **context**
- Generates **problems** than **solution**
- Go from problem to bad solution
- Antipattern consists of two solutions
  - One which is beneficial
  - One which is problematic

Why learning antipatterns?

"Identifying bad practices can be as valuable as identifying good practices" - Ward Cunningham

Antipatterns identify and categorize the common mistakes in software practice

"If debugging is the process of removing bugs, then programming must be the process of putting them in." – Edsger W. Dijkstra

SQL Antipatterns

Database Design

Database Creation

```
CREATE TABLE BugsProducts
(
bug_id INTEGER REFERENCES Bugs,
product VARCHAR(100) REFERENCES
Products,
PRIMARY KEY (bug_id, product)
);
```

Query

```
SELECT b.product, COUNT(*)
FROM BugsProducts AS b
GROUP BY b.product;
```

Application

```
SELECT b.product, COUNT(*)
FROM BugsProducts AS b
GROUP BY b.product;
```

SQL Antipatterns

Design | Creation | Query | Application

Jaywalking

Cross walking

Jaywalking

Example Problem

Storing list of authors for a book…

Book Name	Authors
Code Complete	Steve McConnell
Design Patterns	Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
Intro. to Automata Theory	Jeffrey Ullman, John Hopcroft

Assigning role based menu access…

Menu Name	Role Accessibility
Reset Password	Admin
Account Heads	Senior Accountant, Accounts Manager
YTD Sales Report	CFO, Sales Manager, Sales Rep

SQL Antipatterns

Design | Creation | Query | Application

Jaywalking

Objective

1. Avoid Intersection table

2. Avoid joins and thus minimize query plan complexity

SQL Antipatterns

Design | Creation | Query | Application

Jaywalking

Implementation

```
CREATE TABLE Menu
(
          MenuID INT,
          Menu VARCHAR(50),
          Module VARCHAR(50),
          AccessibileTo VARCHAR(500),

          CONSTRAINT PK_Menu
                    PRIMARY KEY CLUSTERED
          (
                    MenuID
          )
)
GO
```

```
INSERT INTO Menu (MenuID, Menu, Module, AccessibileTo) VALUES
          (1, 'Reset Password', 'Administration', 'Admin'),
          (2, 'Account Heads', 'Finance', 'Senior Accountant, Accounts Manager'),
          (3, 'YTD Sales Report', 'Report', 'CFO, Sales Manager, Sales Rep');
```

SQL Antipatterns

Design | Creation | Query | Application

Jaywalking

New problems…

Violation of First normal form

Cannot chose a right data type – You can enter Admin, Sales Rep, 1, 2, banana

Cannot use foreign key constraints – No referential integrity

Cannot enforce uniqueness – 1, 2, 3, 4, 3, 3, 2

Cannot easily modify the values in the column

What happens if the length of values exceeds the column limit?

How do you search for menus accessible for Admin and CFO? No indexing is possible

How many roles have access to Current Year Budget Reports? Aggregation is difficult

How do you join the table with Role table?

Storing integer number as character take space than storing as integer?

Workarounds include splitting functions that will be inefficient?

SQL Antipatterns

Design | Creation | Query | Application

Jaywalking

Solution : Create an intersection table

Intersection table

```
CREATE TABLE dbo.Menu
(
   MenuID INT,
   Menu VARCHAR(50),
   Module VARCHAR(50)

   CONSTRAINT PK_Menu PRIMARY KEY
     CLUSTERED
     (
            MenuID
     )
)
GO
```

```
CREATE TABLE dbo.MenuAccess
(
   MenuAccessID INT,
   MenuID INT,
   RoleID INT,

   CONSTRAINT FK_MenuAccess_Menu
     FOREIGN KEY (MenuID)
     REFERENCES Menu (MenuID),

   CONSTRAINT FK_MenuAccess_Role
     FOREIGN KEY (RoleID)
     REFERENCES dbo.Role (RoleID)
)
GO
```

```
CREATE TABLE dbo.Role
(
   RoleID INT,
   Name VARCHAR(50),
   CreatedDate DATETIME,

   CONSTRAINT PK_Role PRIMARY KEY
     CLUSTERED
     (
            RoleID
     )
)
GO
```

SQL Antipatterns

Design | Creation | Query | Application

Keyless Entry

Example Problem

I need a Supplier and a Invoice table…

Invoice table stores number from Supplier table, to identify the supplier; however…..

I don’t want to make the design complex….
The sentence I’m about to tell you is a secret ! …

I don’t like keys ! Keys and Constraints limit flexibility!

SQL Antipatterns

Design | Creation | Query | Application

Keyless Entry

Objective

1. Avoid Update/Delete/Insert conflicts

2. Avoid complexity around foreign key constraints

SQL Antipatterns

Design | Creation | Query | Application

Keyless Entry

Implementation

Create tables without any FOREIGN KEY constraints

SQL Antipatterns

Design | Creation | Query | Application

Keyless Entry

New problems…

Breaking the foundation of relational database - Constraints

Introduce meaningless data – Authors without any books or Orders without customers

Cannot utilize the query optimizations due to constraints – Some RDBMS utilizes FOREIGN KEY and CHECK constraints to optimize queries

Need to implement custom solutions to check integrity on a later stage

Forced to implement periodic checks to find orphan rows

SQL Antipatterns

Design | Creation | Query | Application

Keyless Entry

Solution : Implement referential integrity through FOREIGN KEY constraints

Use cascading referential integrity constraints.

```
CREATE TABLE dbo.MenuAccess
(
   MenuAccessID INT,
   MenuID INT,
   RoleID INT,

   CONSTRAINT FK_MenuAccess_Menu
   FOREIGN KEY (MenuID)
      REFERENCES dbo.Menu (MenuID)
       ON DELETE CASCADE
       ON UPDATE CASCADE,

    CONSTRAINT FK_MenuAccess_Role
    FOREIGN KEY (RoleID)
      REFERENCES dbo.Role (RoleID)
        ON DELETE SET NULL
        ON UPDATE CASCADE,
)
GO
```

You can even disable and enable FOREIGN KEY, if needed.

```
-- Disable FOREIGN KEY
ALTER TABLE MenuAccess
NOCHECK CONSTRAINT FK_MenuAccess_Menu;

-- Enable FOREIGN KEY
ALTER TABLE MenuAccess
WITH CHECK
CHECK CONSTRAINT FK_MenuAccess_Menu;
```

SQL Antipatterns

Design | Creation | Query | Application

‘31 Flavors’

Example Problem

Our bug tracking software supports only three bug statuses

NEW
INPROGRESS
FIXED

This will never change (I guess!)

SQL Antipatterns

Design | Creation | Query | Application

‘31 Flavors’

Objective

1. Restrict a column’s values to a fixed set of values
2. Column never contains an invalid entry
3. Simplifies usage and query development

SQL Antipatterns

Design | Creation | Query | Application

‘31 Flavors’

Implementation

```
CREATE TABLE dbo.Bugs
(
   BugID INT,
   ModuleID INT,
   Description VARCHAR(200),

   -- Other columns
   BugStatus VARCHAR(20),

   CONSTRAINT CHK_BugStatus
      CHECK (BugStatus IN
('NEW' , 'IN PROGRESS' , 'FIXED'))
);
GO
```

SQL Antipatterns

Design | Creation | Query | Application

‘31 Flavors’

New problems…

What are the available status values?

```
SELECT DISTINCT BugStatus FROM dbo.Bugs;
```

After few months, the QA team decides to add a new status ‘Duplicate’. How do you do that?

Later the team has instructed to change ‘**Fixed**’ status to ‘**Resolved**’

SQL Antipatterns

Design | Creation | Query | Application

'31 Flavors'

Solution : Create a lookup table and use DRI

```
CREATE TABLE dbo.BugStatus
(
   BugStatusID INT,
   Status VARCHAR(20),
   Description VARCHAR(100),

   CONSTRAINT PK_BugStatus
    PRIMARY KEY CLUSTERED
    (BugStatusID)
)
GO
```

```
CREATE TABLE dbo.Bugs
(
   -- Other columns
   BugStatusID INT,

   CONSTRAINT FK_Bugs_BugStatus
    FOREIGN KEY (BugStatusID)
    REFERENCES dbo.BugStatus
     (BugStatusID)
);
GO
```

```
INSERT INTO dbo.BugStatus (BugStatusID, Status, Description) VALUES
          (1, 'NEW', 'Once again a new bug reported'),
          (2, 'IN PROGRESS', 'Team is working hard on this'),
          (3, 'FIXED', 'The issues fixed for the time being!');
```

SQL Antipatterns

Design | Creation | Query | Application

Fear of Unknown

"There are things known, and there are things unknown, And in between are the Doors." ~ Jim Morrison

Example Problem

Our library stores information on books and periodicals

Both have a name, publisher, language, and number of pages

However ISBN is applicable to books and ISSN is for periodicals.

SQL Antipatterns

Design | Creation | Query | Application

Fear of Unknown

Objective

Storing values that are not available or not applicable

SQL Antipatterns

Design | Creation | Query | Application

Fear of Unknown

Implementation

```
CREATE TABLE dbo.Collection
(
   ID                   INT NOT NULL,
   Name                 VARCHAR(100) NOT NULL,
   Year                 INT NOT NULL,
   PublisherID          INT NOT NULL,

   -- Applicable to Books
   Edition              INT NULL,
   ISBN                 VARCHAR(20) NULL,
   Binding              VARCHAR(20) NULL
     CHECK
            Binding IN ('HARDCOVER', 'PAPERBACK')),

   -- Applicable to periodicals
   FrequencyID          INT NULL
      REFERENCES dbo.Frequency (FrequencyID),

   Volume               INT NULL,
   Issue                INT NULL,
   ISSN                 VARCHAR(10) NULL
)
GO
```

SQL Antipatterns

Design | Creation | Query | Application

Fear of Unknown

New problems...

ID	Name	Year	PublisherID	Edition	ISBN	Binding	FrequencyID	Volume	Issue	ISSN
1	Introduction to Algorithms	1990	2	3	978-0-262-03384-8	PAPERBACK				
2	Code Complete	1993	5	1	978-1-55615-484-3	PAPERBACK				
3	Dr. Dobb's Journal	2009	12				3	34	2	1044-789X
4	The C Programming Language	1978	1	1	0-262-51087-2	PAPERBACK				
5	SQL Server Pro	1999	22				3	7	3	1522-2187

What is the result of

```
SELECT ID FROM dbo.Collection WHERE FrequencyID != 3;?
```

[redacted]

What is the result of

```
SELECT
        Name + ',' + Edition + '(' + ISSN + ')'
FROM dbo.Collection WHERE ID = 2;
```

[redacted]

What is the result of

```
SELECT COUNT(Volume) FROM dbo.Collection;?
```

[redacted]

SQL Antipatterns

Design | Creation | Query | Application

Fear of Unknown

Understanding NULL

… in expression

Expression	Expected	Actual
NULL = 0	TRUE	Unknown
NULL = 12345	FALSE	Unknown
NULL <> 12345	TRUE	Unknown
NULL + 12345	12345	Unknown
NULL + ‘ABC’	ABC	Unknown
NULL = NULL	TRUE	Unknown
NULL <> NULL	FALSE	Unknown

… in Boolean expression

Expression	Expected	Actual
NULL AND TRUE	FALSE	Unknown
NULL AND FALSE	FALSE	FALSE
NULL OR FALSE	FALSE	Unknown
NULL OR TRUE	TRUE	TRUE
NOT (NULL)	TRUE	Unknown

SQL Antipatterns

Design | Creation | Query | Application

Fear of Unknown

Understanding NULL

NULL is not a value in a column

NULL is a marker/construct in SQL for representing missing information

"...it follows that a “type” that “contains a null” isn’t a type,
a “tuple” that “contains a null” isn’t a tuple,
a “relation” that “contains a null” isn’t a relation, and
a “relvar” that contains a null isn’t a relvar.
It follows further that nulls do serious violence to the relational model, and this dictionary therefore has very little to say regarding most null-related concepts“
– C . J. Date – The Relational Database Dictionary

SQL Antipatterns

Design | Creation | Query | Application

Fear of Unknown

Solution: Avoid usage of NULL as far as possible

There are three reasons for NULL creep in to database table…

Inapplicable NULL	*Not yet applicable NULL*	*Nice to know NULL*
The ISSN of periodical is inapplicable to a book **Solution:** Design specialized tables with applicable columns. Create a Book table and a Periodical table Inapplicable NULL should be avoided	The Number of copies of the book is currently not available, but will soon be available **Solution:** Give a default value until the value is available e.g.: ISO scheme for sex: 0 = UNKNOWN 1 = MALE 2 = FEMALE 9 = NOT APPLICABLE	The comment for a book has no business value and will only used for reporting **Solution:** This column can contain the marker NULL.

SQL Antipatterns

Design | Creation | Query | Application

Pseudo Key Neat-Freak

...Do you see the pattern...?

1	2	3
4	5	6
7	8	9
10	11	12

Example Problem

I used auto generated pseudo key for `BookID` in `Book` table

Today morning when I query the table I found gaps in `BookID` numbers...!

Books are missing!

How the database miss it!

Is that a bug!

Clean it up and make it sequential! **Born of a Pseudo Key Neat-Freak**

SQL Antipatterns

Design | Creation | Query | Application

Pseudo Key Neat-Freak

Objective

1. Pseudo key numbers should always be in sequential order
2. There should not be any gaps between two adjacent pseudo key numbers

SQL Antipatterns

Design | Creation | Query | Application

Pseudo Key Neat-Freak

Implementation

1. Do not use auto generating numbers; instead write custom queries to find the next number for the table
2. Periodically run maintenance scripts to fill the gaps

Gaps and Islands – Itzik Ben-Gan (MVP Deep Dives Vol. 1)

Table 3 Performance summary of solutions to gaps problem

Solution	Runtime in seconds	Logical reads
Solution 1—using subqueries	8	62,262
Solution 2—using subqueries	48	31,875,478
Solution 3—using ranking functions	24	32,246
Solution 4—using cursors	250	16,123

10,000,000 rows with 10,000 gaps in every 1,000 rows

SQL Antipatterns

Design | Creation | Query | Application

Pseudo Key Neat-Freak

New problems…

Any custom solution will cause locking and concurrency issues in application

Gap filling algorithms are time consuming and resource intensive

What about missed the propagation of renumbered ID to child tables? You finally introduce ‘Keyless Entry’ Antipattern. Think about a query that returns “Art of computer programming Vol 1” by Chetan Bhagat by joining `Book` and `Author` tables

Nightmare starts if almost all tables have pseudo keys

SQL Antipatterns

Design | Creation | Query | Application

Pseudo Key Neat-Freak

Solution

Use natural keys as far as possible

Get over it: Don't try to renumber a pseudo key. Train the mind to take the 'Pseudo Key' as 'Pseudo'; it has no business value

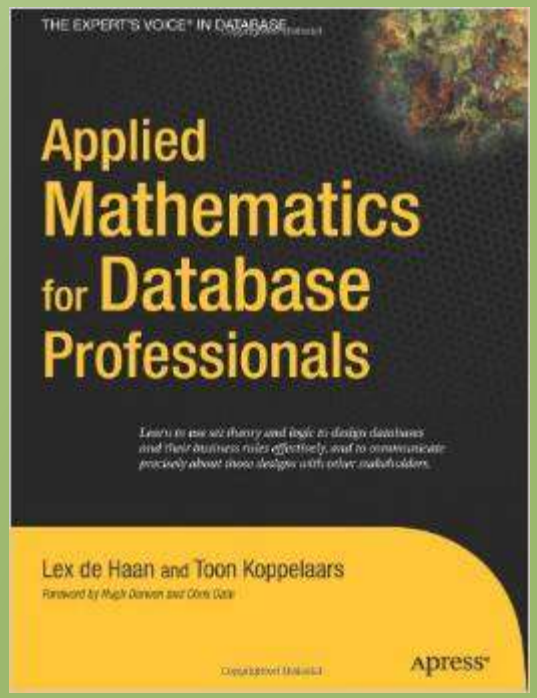
> Another argument for reusing the numbers:
> "After three years the ID will reach the maximum number! I don't want to waste values…"
>
> BIGINT (8 Byte), $2^{63} - 1$ = Max. value 9,223,372,036,854,775,807
> 1 Day = 86400 Seconds
> 1,000 insertions/sec = 106751991167 days ~ **292,471,208 Years**

Note: Pseudo keys are row identifiers; not row numbers

SQL Antipatterns

References

SQL Antipatterns – Bill Karwin



Applied Mathematics for Database Professional – Lex de Haan & Toon Koppelaars

Questions?

Thank You