ARCHIVING DATA

How to Identify Requirements and Implement Solutions

By: CJ Estel, Experity Health

AGENDA

Introduction

Reasons to Archive

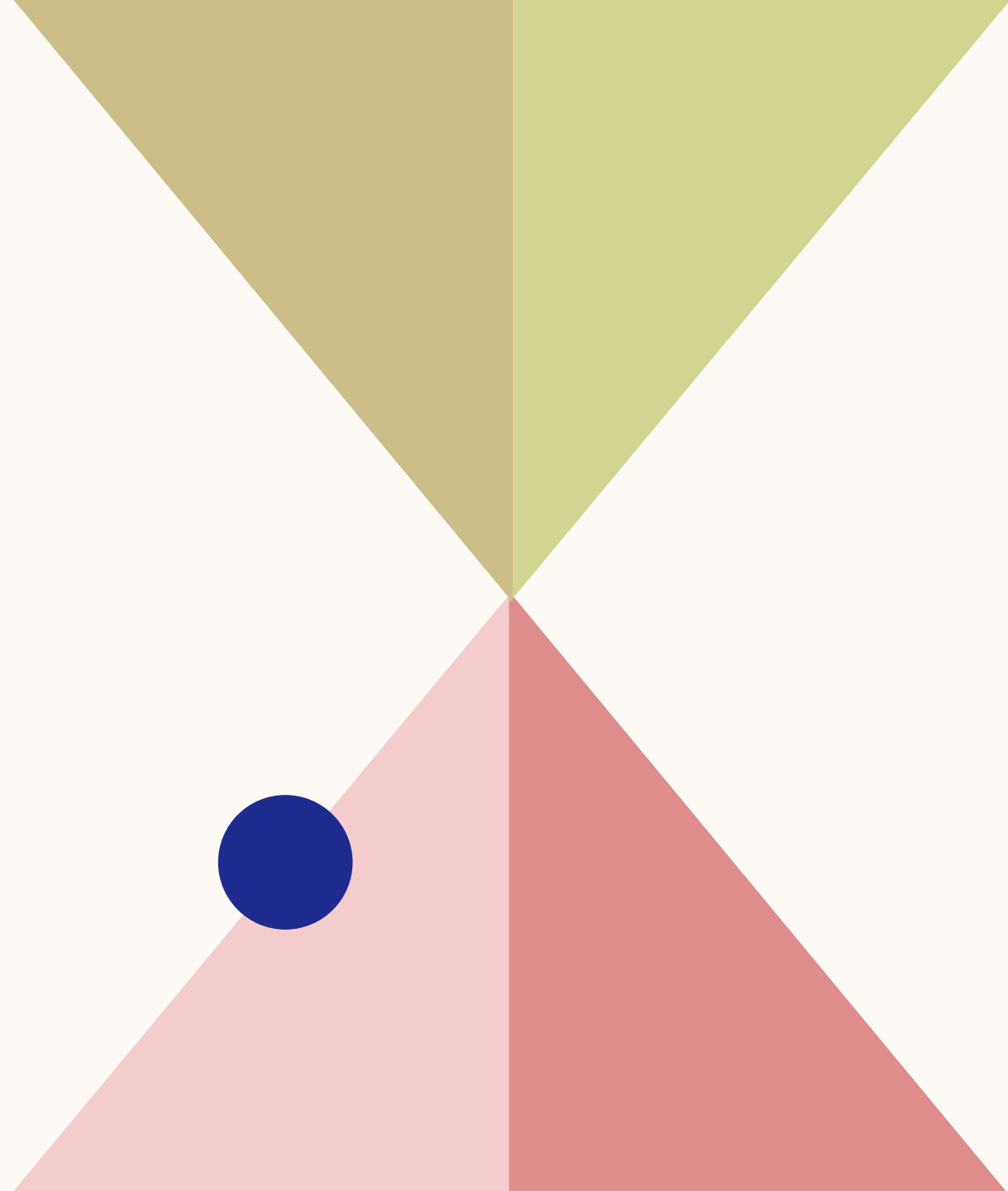
Developing Scope and Requirements

Common Strategies

Our Implementation

Things to Look out For

Code Examples



INTRODUCTION

Who am I?

CJ Estel

What experience do I have?

I have worked with Postgres and MySQL databases for over 20 years. I specialize in High Availability and building Infrastructure as Code (IaC). I have written numerous published puppet modules, deployment tools, and scripts designed to make database administration scalable in large scale environments.

REASONS TO ARCHIVE

- Contractual obligations
- Government regulations
- Makes database tables smaller*and maintenance operations easier to work with
- Makes databases smaller and easier to fail over, backup, etc.
- Can reduce exposure in a data breach
- Can reduce expenses associated with subpoenas for data
- Can speed up queries

DEVELOPING SCOPE AND REQUIREMENTS

QUESTION EVERYTHING

- What is the database archival retention period
- How frequently should archive run
- What is the overall archive retention policy
- What tables does the archival pertain to
- Is the archival based on top level data or referencing tables
- What fields are we archiving based on
- Do any of those archive fields allow for nulls
- What are our maintenance windows
- How quickly would you need an archive restored
- How do you deploy DDL Changes
- Does summary data need preserved
- Do you have any implied foreign keys in code that are not in the database
- Can I access a full copy of production for testing and timing purposes
- Can an application stack connect to this copy of production for app testing (think PHI/sensitive data restrictions
- Are there any tables that you expect to have bad data (future dated, implied foreign key cleanup)
- Are there any upstream or downstream services that may be impacted
- If data flows to analytics, should analytics also delete
- Will your app break if I add or modify tables directly in the database

ARCHIVE DATA ACCESS

- Hot Data Access
  - Consider keeping as detached partition or in an archive schema for period of time
- Warm Data Access
  - Data is no longer in production database but lives easily accessible and loaded on another server (analytics, reporting, etc)
- Cold Data Access
  - Generally held in long term storage, needs restored to use
- No Data Access
  - Data is deleted and not held anywhere

COMMON STRATEGIES

- Partition tables and prune (detach/drop) the partitions as they age out
- Prune Data
- Define Archive Data
  - Hot Data Access
  - Warm Data Access
  - Cold Data Access
  - No Data Access (Delete)

THINGS TO LOOK OUT FOR

- Truly defining what falls into archive period. For Instance
  - are customers a one time thing and we can delete all customers past a certain created_on date?
  - If a customer has a survey within our archive period, do we preserve the customer in that instance?
  - If a survey has some sort of internal tracking that could happen after the survey comes back where it is worked by staff for improvements, we would have newer data in survey feedback associated with an older survey. Do we delete all feedback for surveys older than X, or do we preserve surveys that have feedback that have been worked within our archive period?

    customer
      survey
        survey_feedback

THINGS TO LOOK OUT FOR (CONT)

- On Delete cascade can make things WAY simpler to delete children objects, but it can also make it WAY harder in the event you ever need to restore data.
- Bad data, implied foreign keys, and implied constraints should be corrected and formalized in the database.
- If considering partitions
  - Understand run times to convert to table (and workarounds/risks of online methods)
  - Understand restrictions on uniqueness and need for check constraints on unique fields
  - Understand foreign keys to the table and their potential impacts on detaching partitions
  - Automate your future partition creation (pg_partman)

THINGS TO LOOK OUT FOR (CONT)

- Will replicas, analytics, etc be impacted by the methods you use to adjust ddl (example: data isn't archived in analytics)
- If using something like pg_partman, will application ORM crash if changes to objects are detected
- Will your archive introduce differences between environments and if so, how are those tracked/deployed consistently
- Will the archive cause backups in replication (trigger load, queues, cdc, etc)

- You will likely want to rebuild indexes after archival

OUR IMPLEMENTATION

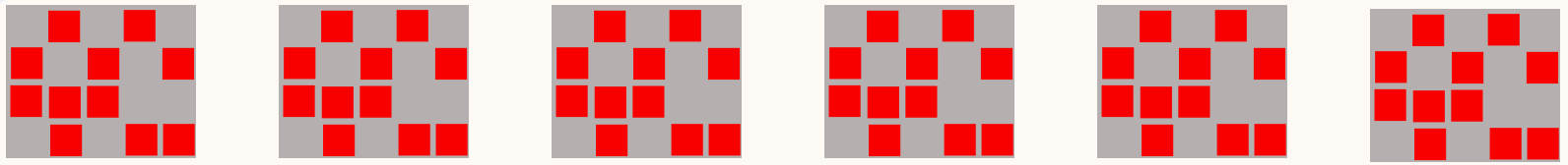
- Identify Scope
- Cleanup data (Implement not null, remove bad data, implement missing foreign keys, etc)
- Based on foreign key dependencies, define the order in which you will archive the tables data
- Implement any indexes on fields that will be necessary to facilitate queries for archiving
- Identify if table will be archived using partition or pruning method

OUR IMPLEMENTATION (CONT)

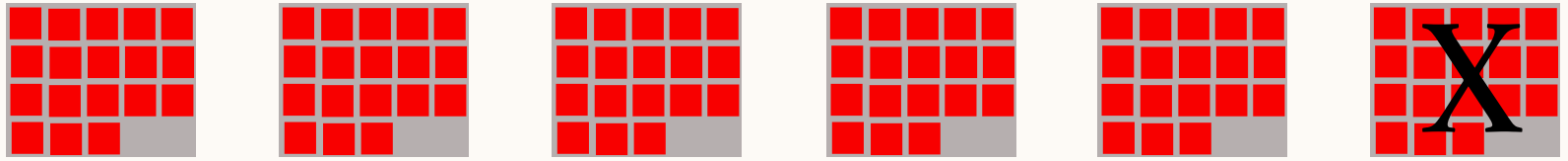
- If pruning, identify select queries that will appropriately select the data to be archived starting with the child most tables
- Delete/move these records from production table
  - ideally using a function or wrapper that can delete rows in batches, example to come
- If holding as hot data set, have processes that will cycle the hot data to files that can be moved to long term storage

*HOW MUCH DISK SPACE WILL I SAVE

- Pruning Data



- Pruning Partitions



FOLLOW UP STEPS

- Rebuild all indexes for the table
- Consider manually kicking off a vacuum
- Consider repacking your data
- If using cold long term storage for files, ensure that storage has retention rules

CODE EXAMPLES

QUESTIONS?

CJ Estel

https://github.com/cjestel/databases

https://github.com/cjestel/conferences