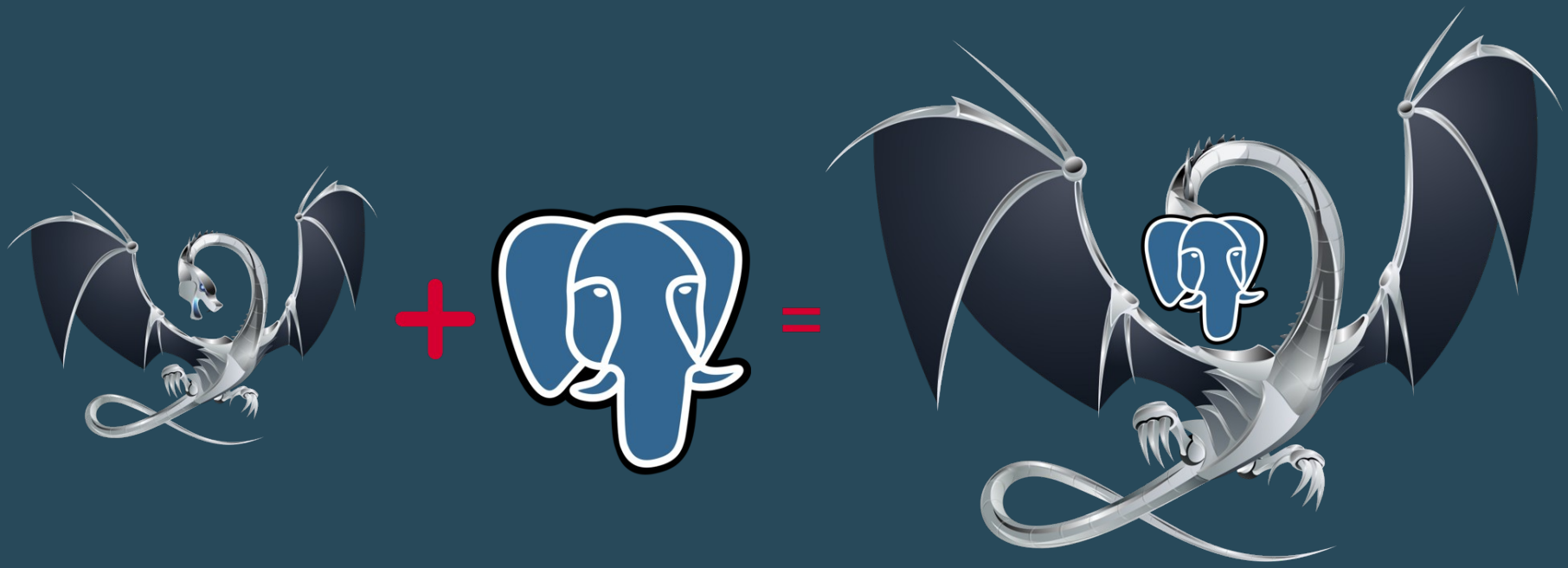
EDB

JIT no PostgreSQL: aliado ou não?



Juliano Atanazio

Como uma consulta é executada

Conexão → Parser → Rewriter → Planner → Executor

Conexão

Uma aplicação se conecta a uma instância de banco de dados Postgres e então a partir disso se pode transmitir consultas.

Como uma consulta é executada

Conexão → Parser → Rewriter → Planner → Executor

Parser

É o estágio que faz a análise sintática da consulta transmitida pela aplicação e então cria uma árvore de consulta.

Como uma consulta é executada

Conexão → Parser → Rewriter → Planner → Executor

Rewriter

A partir da árvore de consulta criada pelo *Parser, o Rewriter* procura por quaisquer regras que estão armazenadas em catálogos e então aplicar a essa árvore.

Transformações são feitas de acordo com os corpos das regras.

Por exemplo, ao se consultar uma *view*, a consulta é reescrita de maneira a acessar diretamente a(s) tabela(s) de sua consulta de construção.

Como uma consulta é executada

Conexão → Parser → Rewriter → Planner → Executor

Planner / Optimizer

Determina qual a melhor forma de execução da consulta baseando-se em custos.

Como uma consulta é executada

Conexão → Parser → Rewriter → Planner → Executor

Executor

A partir do caminho entregue pelo *Planner* efetivamente executa a consulta.

Ele percorre o plano de execução gerado *Planner*, que significa:

- Avaliar expressões (`WHERE`, `JOIN`, `HAVING`, etc);
- Execução de funções de agregação;
- Processar filtros e projeções;
- Loop sobre linhas vindas de buscas.

Como uma consulta é executada

Conexão → Parser → Rewriter → Planner → Executor

Executor

```
SELECT
    sum(valor)
FROM vendas
WHERE dt_venda >= '2025-01-01';
```

- `WHERE` → filtros;
- `sum` → agregação;
- `FROM` → scan;
- `dt_venda >= '2025-01-01'` → expressão a ser compilada pelo JIT.

Just in time

A expressão *just in time* é aplicada de acordo com um contexto, porém mantendo a ideia de tempo.

• **Bem na hora**: algo acontecendo no momento necessário;

• **Produção sob demanda**: contexto logístico; em vez de estocar produzir conforme pedidos;

• **Em tempo real**: no âmbito computacional significa processamento ou execução feitos exatamente no momento necessário, sem etapas antecipadas desnecessárias.

Sobre o recurso JIT compilation

Just-In-Time compilation ou compilação em tempo de execução no PostgreSQL é um recurso que compila partes do plano de execução de uma consulta em código de máquina enquanto a consulta é executada em vez de antes da execução.

Com o JIT há uma substituição de partes interpretadas por código de máquina otimizado, tais como expressões lógicas e matemáticas, condições, agregações pesadas e *loops*.

https://www.postgresql.org/docs/current/jit-reason.html

Sobre o recurso JIT compilation

O PostgreSQL tem suporte a compilação JIT utilizando LLVM (https://llvm.org) quando em sua compilação utilizar a opção `--with-llvm`.

LLVM *(Low Level Virtual Machine*; máquina virtual de baixo nível*)* é um conjunto de ferramentas e uma infraestrutura modular para compiladores e otimizadores de código, inicialmente projetado para fornecer compilação *just in time* e otimizações em tempo de compilação, linkagem e execução.

Mais informações podem ser obtidas no código-fonte do PostgreSQL em:
`src/backend/jit/README`

JIT no PostgreSQL: motivação

O JIT no PostgreSQL começou na versão 11, lançada em Outubro de 2018.

A principal razão para introduzir o JIT no PostgreSQL foi para melhorar a performance de execução de consultas, especialmente para expressões complexas e operações.

Antes do JIT, o PostgreSQL interpretava essas operações, o que gerava *overhead*.

Com o JIT, expressões são compiladas para o código nativo da máquina em tempo de execução, permitindo o código ser executado diretamente na CPU, eliminando interpretação e otimizando o processo.

Quando NÃO usar o JIT

É importante saber que embora o JIT possa aumentar a velocidade de execução de forma significativa, há o tempo de compilação.

Para consultas simples e de curta duração, pode não ser uma boa ideia, devido ao tempo de compilação gasto pode não compensar os benefícios de desempenho na execução.

Por esse motivo, há determinados parâmetros para configurar quando é melhor utilizar o recurso de JIT dada uma consulta.

Limitações

Overhead de compilação

Não recomendável para consultas simples, de curta duração ou que sejam executadas raramente.

O PostgreSQL tenta mitigar isso usando limites de custo, como o parâmetro `jit_above_cost` para decidir quando vale a pena usar o JIT, porém a decisão é baseada em estimativa de tempo e planejamento.

Limitações

Não otimiza todas as partes da consulta

Expressões e avaliação de tuplas é onde o JIT se concentra principalmente, além de algumas operações internas como agregações.

Nem todas as fases da consulta são otimizadas como:

• I/O; quando o gargalo não está na CPU não terá muito efeito;

• *Locks* e concorrência; não são resolvidos pelo JIT;

• Escolha do plano de execução; O JIT é subordinado ao planejador de consultas *(planner)* e não tem como esse recurso determinar o plano.

Limitações

Sem cache de código JIT persistente (atualmente)

O código de máquina gerado pelo JIT não é persistido ou armazenado em *cache* de forma eficaz entre execuções da consulta (ou entre sessões).

Ou seja, não há uma forma global para isso, mas sim por sessão.

Para cada execução em que o JIT é acionado uma nova compilação é feita, o que aumenta o *overhead* para consultas repetidas, embora haja trabalho em andamento nessa área para versões futuras.

Quando usar o JIT

É extremamente aconselhável que sejam feitos testes com `EXPLAIN ANALYZE` previamente para saber se vale a pena habilitar ou não.

Porém há sim casos que o JIT pode ser mais benéfico, os quais serão vistos a seguir.

Quando usar o JIT

Consultas analíticas complexas

São obtidos ganhos de desempenho com o JIT em consultas que envolvem muitas operações como:

- **Funções e expressões complexas**
  - Manipulações de *strings*;
  - Cálculos matemáticos;
  - UDFs* cujas execuções são feitas repetidamente sobre grandes conjuntos de dados;

* UDF: *User-Defined Function*; Função Definida pelo Usuário

Quando usar o JIT

Consultas analíticas complexas

- **Joins com muitas condições**

  Casos que têm várias condições de filtragem ou junção que podem ser otimizadas.

Quando usar o JIT

Consultas analíticas complexas

- **Agregações com funções customizadas**

  Funções de agregação que não são as padrão (e. g: `avg`, `sum`, `count`) e que podem ser compiladas para um código mais eficiente.

Quando usar o JIT

Consultas analíticas complexas

- **Processamento de grandes volumes de dados**

  Quanto mais dados uma consulta processa, maior o potencial de ganho de desempenho que o JIT pode proporcionar, devido ao custo da compilação ser amortizado sobre um maior volume de trabalho.

Quando usar o JIT

Workloads com consultas repetitivas

Quando há um conjunto de consultas complexas que são executadas muitas vezes, o custo inicial da compilação é compensado pelos ganhos de performance em execuções subsequentes.

Isso é devido ao PostgreSQL poder armazenar em *cache* os planos de execução compilados.

Quando usar o JIT

Ambientes com CPU bound (gargalo de processamento)

Quando a CPU é o principal gargalo, o JIT pode reduzir significativamente o tempo de execução da CPU, liberando recursos para outras tarefas.

Quando usar o JIT

Versões mais recentes do PostgreSQL

Como dito anteriormente, o JIT foi um recurso que surgiu no PostgreSQL a partir da versão 11 e a cada versão nova esse recurso tem sido melhorado.

Quanto mais recente for a versão do PostgreSQL mais madura e otimizada será a implementação do JIT.

Como saber se o JIT foi usado

O comando `EXPLAIN (ANALYZE, VERBOSE)` mostra, no final do plano, uma seção "JIT" com:

- Número de funções geradas;
- Tempo de geração;
- Fases como *“Inlining”*, *“Optimization”*, *“Emission”*.

Exemplo:

```
JIT:
   Functions: 4
   Timing: Generation 0.814 ms, Optimization 4.828 ms, Emission 33.978 ms
```

Caso a seção "JIT" esteja ausente, o JIT não foi usado.

JIT tuple deforming

Tuple deforming significa quebrar uma tupla (linha) armazenada em um bloco de dados em suas colunas individuais para serem processadas pelo *executor*.

Tupla armazenada: é compactada, com cabeçalhos, alinhamento e pode ter colunas `NULL`.

Deformar: ler os *bytes* da tupla e materializar os valores das colunas em variáveis em tempo de execução.

Quando este recurso está habilitado (`jit_tuple_deforming`), o PostgreSQL gera código de máquina otimizado para deformar tuplas, assim evita o uso de *loops* genéricos em C para cada coluna.

JIT tuple deforming

Esse código gerado conhece de antemão o *layout* da tupla e acessa as colunas de forma individual, sem laços genéricos.

Ganhos de performance aparecem em consultas com muitas colunas ou alto volume de linhas.

Parâmetros JIT

jit

Habilita ou não o recurso de JIT.

Parâmetros JIT

jit_above_cost

Define o custo mínimo de uma consulta para utilizar o JIT.

Parâmetros JIT

jit_optimize_above_cost

Limite de custo acima do qual o PostgreSQL aplica otimizações adicionais do LLVM no código JIT gerado.

Caso esse valor seja atingido o PostgreSQL além de gerar o código JIT, vai também rodar otimizações do LLVM (e. g. *inlining* de expressões, eliminação de código morto, entre outras).

Deixar seu valor muito baixo aumentará o tempo de compilação e não compensa para consultas pouco complexas.

Parâmetros JIT

jit_inline_above_cost

Custo mínimo de uma consulta para que o PostgreSQL permita que o LLVM faça *inlining* de funções (e. g.: PL/PgSQL) no código JIT compilado.

Parâmetros JIT

jit_tuple_deforming

Habilita ou não o recurso de *tuple deforming*.

Conclusão

- Não use o JIT indiscriminadamente;
- Ajuste parâmetros conforme a carga;
- Sempre avalie o custo da compilação vs ganho na execução;
- Teste, utilize `EXPLAIN ANALYZE`, analize os planos e ajuste os custos. Os testes devem ter um volume de dados considerável, o mais próximo possível com o que tem em produção;
- Só então decida pelo JIT.