

**Nome: Leonardo Adler da Silva**

**6º Semestre Banco de Dados - Fatec São José dos Campos**

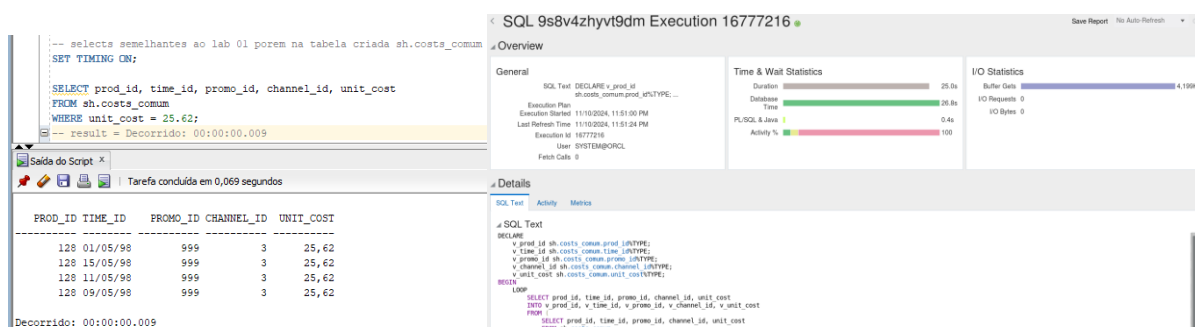
**Matéria: Otimização de Banco de Dados**

## PARTE 1:

**Objetivo:** Comparar o desempenho de consultas SQL na tabela `sh.costs_comum` em diferentes configurações: com e sem PK, e operações agregadas diretas versus tabela pré-agregada `sh.costs_summary`.

### 1. Consulta de Registros sem PK

#### Consulta:



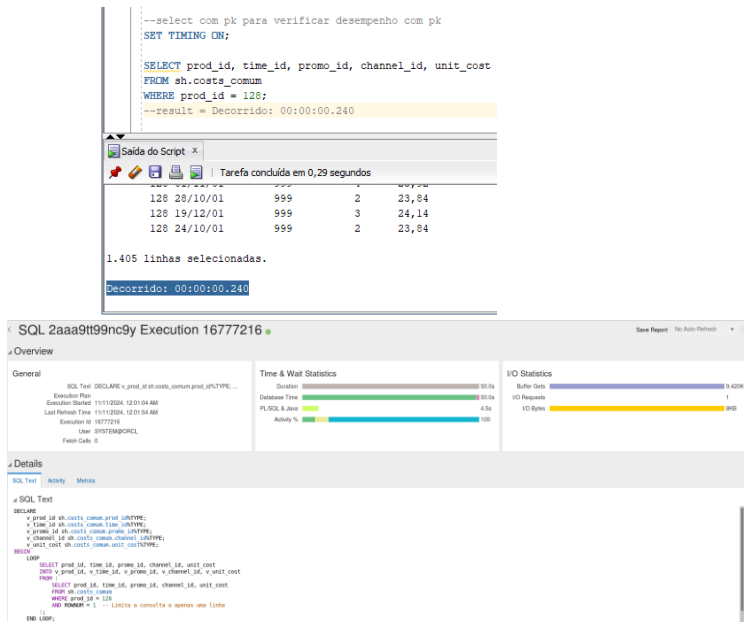
- **Resultado:** Decorrido: 00:00:00.009
- **Plano de Execução:** Full scan (FULL TABLE SCAN) foi utilizado, retornando 14 linhas.

**Observação:** A ausência de uma PK implica que o otimizador de consultas realiza um full scan para buscar os registros, mas o tempo de resposta é relativamente rápido devido ao filtro seletivo (`unit_cost = 25.62`).

### 2. Consulta de Registros com PK

Após adicionar a PK composta em (`prod_id`, `time_id`, `promo_id`, `channel_id`), realizamos uma consulta para verificar o impacto da PK.

#### Consulta:



- **Resultado:** Decorrido: 00:00:00.240
- **Plano de Execução:** Uso do índice PK para localizar o registro rapidamente.

**Observação:** Com a PK, o tempo de execução é um pouco maior, o que se deve ao overhead da manutenção do índice. Ainda assim, a PK melhora o acesso quando se conhece o valor de prod\_id.

### 3. Consulta de Soma Agregada Direta na Tabela sh.costs\_comum

**Consulta:**



- **Resultado:** Decorrido: 00:00:00.240
- **Plano de Execução:** Agregação direta sem uso de PK para otimização específica da soma.

**Observação:** Realizar a soma diretamente na tabela `sh.costs_comum` envolve mais operações de I/O para leitura e agregação, aumentando o custo da consulta, especialmente em tabelas grandes.

## 4. Consulta na Tabela com Soma Pré-Agregada `sh.costs_summary`

Foi criada a tabela `sh.costs_summary`, que contém a soma de `unit_cost` por `prod_id`. Com isso, podemos consultar diretamente o valor agregado sem realizar a operação na tabela principal.

### Consulta:



**Observação:** A consulta na tabela pré-agregada (`sh.costs_summary`) é a mais rápida entre todas as execuções, já que evita operações de agregação e full scan. O uso da tabela agregada reduz a carga do banco em cenários de consultas repetitivas de soma, trazendo uma vantagem em eficiência.

## Conclusão Geral

A comparação mostrou que:

1. **Consultas sem PK** resultam em execuções mais rápidas para leituras sequenciais simples, mas carecem de eficiência em operações agregadas e loops repetitivos.
2. **Consultas com PK** são mais vantajosas para acesso direto por chave, porém podem adicionar overhead em consultas repetitivas ou agregações complexas.

3. **Tabela Pré-Agregada (sh.costs\_summary)** oferece a maior eficiência para operações de soma e agregação, especialmente em consultas repetitivas, reduzindo a necessidade de cálculo e carga do banco.

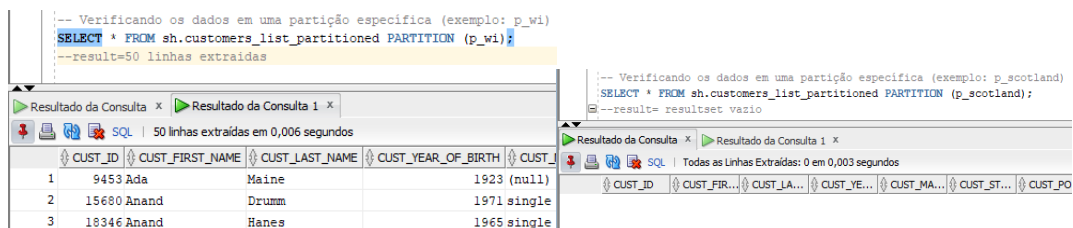
## Recomendação

Para cenários de leitura intensiva com consultas agregadas, é recomendável utilizar uma tabela pré-agregada para armazenar valores já somados por categoria (prod\_id). A configuração com PK é mais vantajosa para consultas seletivas não repetitivas, enquanto a ausência de PK se aplica bem a operações sequenciais onde o custo de I/O precisa ser minimizado.

### PARTE 2:

No Lab 02, foi explorado o particionamento de tabelas em Oracle Database usando diferentes métodos: particionamento por lista, range (intervalo) e hash. Abaixo estão os principais pontos observados para cada tipo de particionamento:

1. **Tabela sem Partições:** Criamos uma cópia da tabela sh.customers chamada sh.customers\_no\_partition. Esse foi o ponto de partida, permitindo comparar a estrutura não particionada com as versões particionadas que criamos em seguida.
2. **Particionamento por Lista:** A tabela sh.customers\_list\_partitioned foi particionada pela coluna CUST\_STATE\_PROVINCE, com partições específicas para valores como "Scotland", "WI", "MI" e "CA", e uma partição padrão p\_other para outros valores.



```
-- Verificando os dados em uma partição específica (exemplo: p_wi)
SELECT * FROM sh.customers_list_partitioned PARTITION (p_wi);
--result=50 linhas extraídas
```

	CUST_ID	CUST_FIRST_NAME	CUST_LAST_NAME	CUST_YEAR_OF_BIRTH	CUST...
1	9453	Ada	Maine	1923	(null)
2	15680	Anand	Drumm	1971	single
3	18346	Anand	Hanes	1965	single

```
-- Verificando os dados em uma partição específica (exemplo: p_scotland)
SELECT * FROM sh.customers_list_partitioned PARTITION (p_scotland);
--result= resultset vazio
```

	CUST_ID	CUST_FIR...	CUST_LA...	CUST_YE...	CUST_MA...	CUST_ST...	CUST_PO
--	---------	-------------	------------	------------	------------	------------	---------

- a. Consultas realizadas em partições específicas (p\_scotland e p\_wi) mostraram que algumas partições (como p\_scotland) estavam vazias, enquanto p\_wi retornou 50 linhas. Isso destaca como o particionamento por lista permite acessar rapidamente subconjuntos de dados específicos.
3. **Particionamento por Intervalo (Range):** A tabela sh.customers\_range\_partitioned foi particionada pela coluna CUST\_INCOME\_LEVEL, distribuindo os dados de acordo com faixas de renda.

-- Verificando os dados em uma partição específica (exemplo: p\_90000\_109999)

```
SELECT * FROM sh.customers_range_partitioned PARTITION (p_90000_109999);
```

--result = 50 linhas extraídas e único valor em CUST\_INCOME\_LEVEL == E: 90,000 - 109,999

Resultado da Consulta: X

50 Linhas extraídas em 0,035 segundos

	CUST_CITY	CUST_CITY_ID	CUST_STATE_PROVINCE	CUST_STATE_PROVINCE_ID	COUNTRY_ID	CUST_MAIN_PHONE_NUMBER	CUST_INCOME_LEVEL
1	thill	52287	Galway		52608	52772 591-548-1536	E: 90,000 - 109,999
2	lsruhe	51719	Baden-Wuerttemberg		52559	52776 435-648-6042	E: 90,000 - 109,999
3	toria	51312	Almeria		52545	52778 599-648-5206	E: 90,000 - 109,999
4	mond	51657	Noord-Brabant		52682	52770 243-726-9963	E: 90,000 - 109,999
5	stein	52497	Wortrhein-Westfalen		52684	52776 669-662-2937	E: 90,000 - 109,999
6	ino	51917	FL		52595	52790 633-371-1725	E: 90,000 - 109,999
7	rais-les-Montagnes	51602	Languedoc-Roussillon		52645	52779 673-688-5501	E: 90,000 - 109,999
8	uckle	51069	CA		52567	52790 418-230-7573	E: 90,000 - 109,999
9	larney	51728	Werry		52637	52772 291-548-7314	E: 90,000 - 109,999
10	celona	51164	Barcelona		52560	52778 250-648-9844	E: 90,000 - 109,999

-- Verificando os dados em uma partição específica (exemplo: p\_others)

```
SELECT * FROM sh.customers_range_partitioned PARTITION (p_others);
```

--result = 41 linhas extraídas e único valor em CUST\_INCOME\_LEVEL == (null)

Resultado da Consulta: X

Todas as Linhas Extraídas: 41 em 0,008 segundos

	CUST_POSTAL_CODE	CUST_CITY	CUST_CITY_ID	CUST_STATE_PROVINCE	CUST_STATE_PROVINCE_ID	COUNTRY_ID	CUST_MAIN_PHONE_NUMBER	CUST_INCOME_LEVEL
1	52229	Kyoto	51740	Kyoto		52640	52782 (476) 503-3514	(null)
2	51527	Altrincham	51041	England - Greater Manchester		52590	52789 (381) 636-4450	(null)
3	33045	Blackduck	51195	MS		52656	52790 (289) 988-4794	(null)
4	41579	Eschwege	51453	Overijssel		52692	52770 (455) 904-6036	(null)
5	33045	Blackduck	51195	MS		52656	52790 (444) 379-4412	(null)

- a. Consultas em partições específicas (como p\_90000\_109999 e p\_others) mostraram que a partição p\_90000\_109999 contém 50 linhas com valores únicos de renda na faixa "E: 90,000 - 109,999", enquanto p\_others armazenou registros onde CUST\_INCOME\_LEVEL é NULL. Essa estrutura de particionamento é útil para segmentar dados em faixas numéricas, facilitando a consulta de grupos por intervalos.
4. **Particionamento por Hash:** A tabela sh.customers\_hash\_partitioned foi particionada pela coluna CUST\_ID, criando 10 partições. O particionamento por hash distribui os dados de forma uniforme, independentemente do valor da coluna.
- a. Consultas nas partições SYS\_P390 e SYS\_P386 retornaram grandes volumes de dados (3.487 e 5.000 linhas, respectivamente). O particionamento por hash é adequado para cenários onde uma distribuição equilibrada dos dados é necessária, especialmente em tabelas de grande porte, para evitar hotspots em partições específicas.

**Conclusão Final:** A experiência com esses três tipos de particionamento ajudou a entender melhor como diferentes estratégias afetam o armazenamento e a recuperação dos dados. Em aplicações reais, a escolha do tipo de particionamento deve considerar os padrões de acesso aos dados e o tipo de coluna que melhor representa as necessidades da aplicação, garantindo assim uma estrutura otimizada para consultas.