Lucas Caltabiano: 112800305

Christoph Califice: 121048060

Descrição do problema

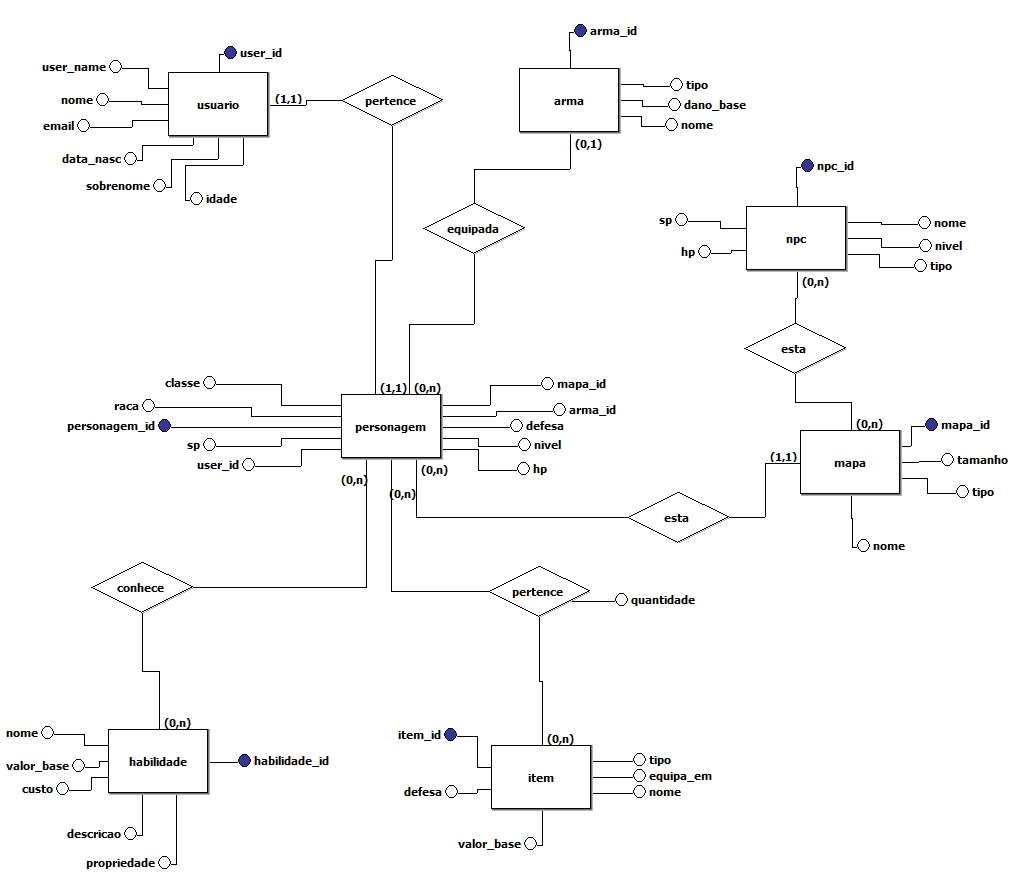
O problema consiste em modelar um sistema simples para ser usado em um jogo de RPG online.

## Conceitos básicos

* Usuário: representa a pessoa que se cadastrou no jogo e contém os dados de identificação do usuário.
* Personagem: é a representação do usuário no jogo. Sua raça e classe são definidas no momento da criação do personagem. Os pontos de vida (hp) e pontos de magia (sp) são calculados com base na classe, raça e nível.
* Arma: somente uma arma pode ser equipada por personagem em um determinado momento.
* Mapa: é o local onde o personagem e os outros elementos do jogo estão localizados. O tipo do mapa determina os tipos de eventos que podem ocorrer.
* Habilidade: são comuns à raças ou classes. Podem ser de diversas propriedades, podendo infligir dano ou modificar status de personagens.
* Item: Encontrados durante o jogo. Alguns podem ser consumidos pelo personagem e outros podem ser equipamentos.
* NPC: non-player character, qualquer elemento que não seja o jogador. Podem ser personagens aliados, inimigos, monstros, etc.

## Regras de negócio

* Usuário:
  + O username deve ser único
  + A idade do jogador deve ser maior que 16 anos. É calculada a partir da data de nascimento por uma trigger e validada por check constraint.
  + O e-mail deve ser validado, contendo ‘@’. Implementado por check constraint.
* Personagem:
  + A classe deve ser uma das seguintes: arqueiro, feiticeiro, templário, trapaceiro. Implementado por check constraint.
  + A raça deve ser umas das seguintes: humano, orc, elfo, dragoniano, goblin. Implementado por check constraint.
  + O HP e o SP do personagem é calculado de acordo com sua classe, raça e nível. Implementado por trigger.
  + Para que o personagem adquira uma nova habilidade, uma trigger verifica se o custo de SP da habilidade não é maior que a capacidade máxima de SP deste.
  + A defesa do personagem é calculada usando uma trigger que verifica os itens atribuídos a este na tabela associativa.
* Arma:
  + O tipo da arma deve ser um dos seguintes: cajado, espada, maça, adaga, arco. Implementado por check constraint.
* Habilidade:
  + Deve ser de uma das seguintes propriedades: fogo, gelo, neutro, vento, terra, status, cura. Implementado por check constraint.
* Item:
  + Itens com quantidade menor ou igual a zero são removidos da tabela associativa usando uma trigger.
* Npc:
  + Deve ser de um dos seguintes tipos: inimigo, aliado, neutro. Implementado por check constraint.
* Mapa
  + Deve ser de uma das categorias: cidade, campo, calabouço. Implementado por check constraint.

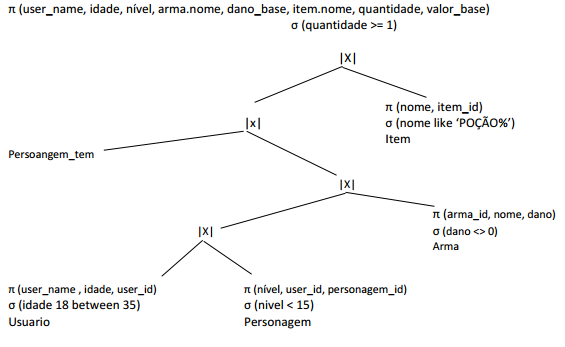


## Consultas

Consulta 1:

Selecionar um resumo dos itens e armas dos personagens que se encaixam nas condições: Idade do usuário entre 18 e 35 anos, nível do personagem menor que 15, armas com dano não nulo e que tenham mais que 1 poção.

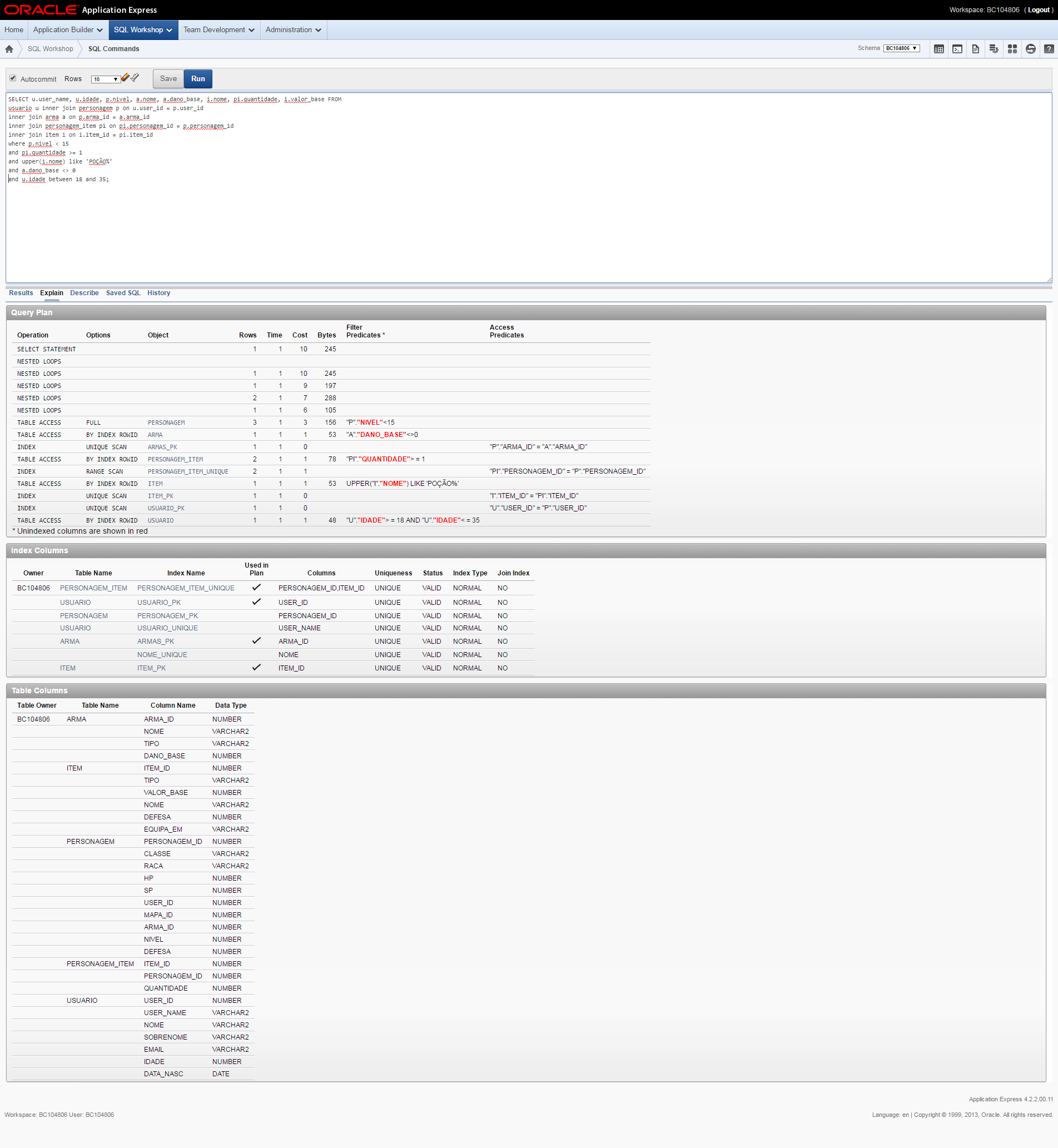
Árvore otimizada por regras:



Resultado da consulta:



Plano de execução do APEX:



Comparação entre os planos de execução:

A primeira operação na nossa árvore de execução é a junção das tabelas de usuário e personagem aplicando os respectivos filtros em cada, porém a primeira operação do plano de execução do Oracle faz a junção das tabelas Personagem e Arma, fazendo uma leitura completa da tabela de personagens e depois é aplicado o filtro por nível. Na tabela Arma é primeiro realizada a junção com a tabela de personagens e em seguida é aplicado o filtro pelo dano base da arma.

Em seguida, na nossa árvore, realizamos a junção com a tabela Arma, enquanto o Oracle realiza a junção com a tabela associativa de personagens e itens. Esta operação é executada em seguida no nosso plano, mas o Oracle neste momento faz junção com a tabela de itens.

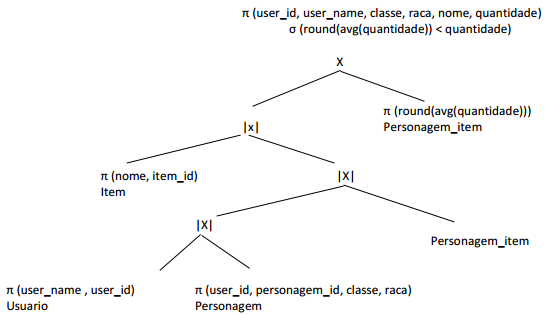
Por último, a junção final no nosso plano é com a tabela de itens o Oracle realiza a junção com a tabela de usuários.

Opinião sobre a otimização:

Percebemos que na operação de acesso aos dados da tabela Personagem foi usada a opção *Full Access*, provavelmente pelo fato de haver poucos registros inseridos nas tabelas. Quando elaboramos a nossa árvore, tínhamos em mente um cenário em que haveria um número maior de registros inseridos no banco, de modo que filtrando os registros antes de combiná-los seria mais eficiente. Como no cenário de execução das consultas não haviam muitos dados inseridos, o plano de execução do Oracle usou outras operações que seriam mais eficientes pare este cenário.

Consulta 2:

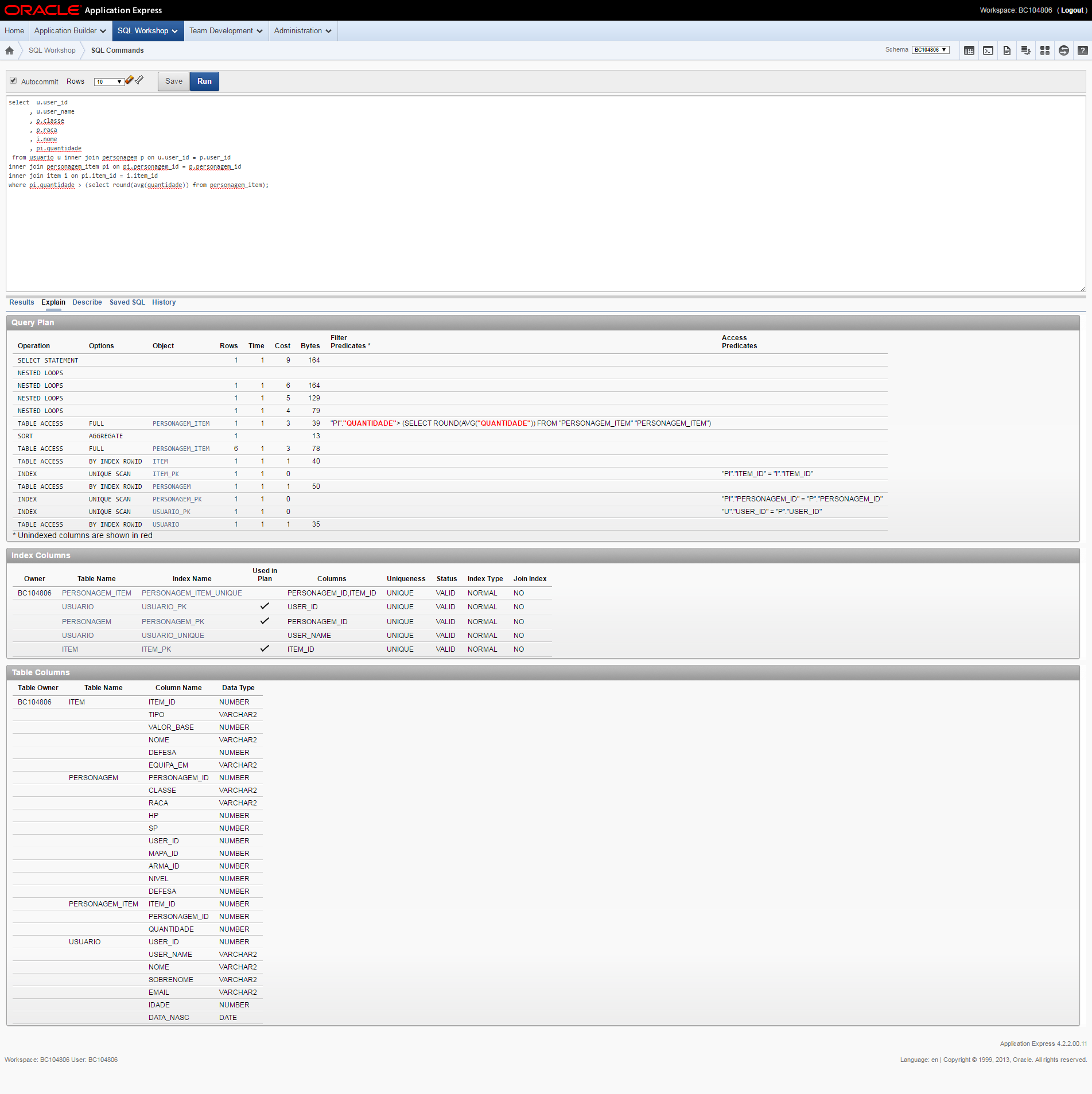
Um resumo das informações de usuário e seu personagem (user\_id, user\_name, classe, raca, nome) que tem mais itens do que a quantidade média de itens dos personagens.



Resultado da consulta:



Plano de execução do APEX:



Comparação entre os planos de execução

O Oracle inicialmente faz a função de agregação através de um full access na tabela Personagem\_Item, em nossa árvore, deixamos esse acesso por último.

Na nossa árvore, iniciamos primeiramente com consultas e filtros na tabela Usuário e Personagem, seguida da junção entre essas tabelas e juntando o resultado disso com o resultado da consulta já filtrada na tabela Arma.

O Oracle no entanto, nesse momento realiza outro full access na tabela personagem\_item, seguido de uma junção com a tabela Item e deste ponto em diante, faz a junção com Personagem e depois Usuário. Em nossa versão da árvore, a consulta na tabela Item é o último passo realizado.

Opinião sobre a otimização:

Percebemos que na operação de acesso aos dados da tabela Personagem foi usada a opção *Full Access*, provavelmente pelo fato de haver poucos registros inseridos nas tabelas. Quando elaboramos a nossa árvore, tínhamos em mente um cenário em que haveria um número maior de registros inseridos no banco, de modo que filtrando os registros antes de combiná-los seria mais eficiente. Como no cenário de execução das consultas não haviam muitos dados inseridos, o plano de execução do Oracle usou outras operações que seriam mais eficientes pare este cenário.