4
Implementação

No Capítulo 3 foi apresentada a proposta de Tese, que diz respeito a definição de um modelo conceitual da biologia molecular e estrutura de armazenamento e acesso a dados de sequências biológicas. Foi definida a utilização do tipo de dados *string* para armazenamento de sequências biológicas e para acesso e manipulação foram definas funções específicas ao contexto/domínio da biologia genômica.

Neste Capítulo é apresentado a implementação do tipo abstrato de dado proposto, bem como algumas consultas sobre um esquema lógico elaborado a partir do modelo conceitual biológico proposto. O objetivo é evidenciar a expressividade do modelo com relação à gestão e análise de dados biológicos. Além disso, é apresentado como as funções propostas para o tipo abstrato de dado foram implementadas, descrevendo qual a lógica empregada e estruturas auxiliares de armazenamento utilizadas.

Para a implementação e teste dos conceitos empregados e o uso das funções propostas, foi realizado um estudo de caso utilizando o SGBD PostgreSQL [PostgreSQL 2012], na sua versão 9.1, para prototipação das soluções apresentadas.

4.1.
Modelagem, Armazenamento e Acesso de Sequências Biológicas

4.1.1.
Armazenamento de Sequências Biológicas

Conforme apresentado nos Capítulos anteriores, não há na literatura, tão pouco implementado por um SGBD, uma estrutura de dados dedicada a sequências biológicas. Atualmente este tipo de dado é tratado como um dado binário (BLOB) ou como um conjunto de caracteres (*string*) *core*.

Neste primeiro momento, não entraremos no mérito de como os SGBDs armazenam tais estruturas. Nosso foco está inserido na forma como o dado é manipulado. Não existem mecanismos adequados para manipular ou extrair informação de domínio biológico de sequências biológicas.

4.1.2. Manipulação de Sequência Biológia

A grande questão desta Tese é a ausência de mecanismos de acesso específicos a dados biológicos, quando utilizado um SGBD comercial. Para exemplificar, caso seja utilizado o tipo de dados TEXT para armazenar e gerenciar sequências biológicas dispomos apenas de um conjunto de funções destinadas a manipulação de texto (*string*).

A seguir são descritas as funções propostas nesta Tese para acesso e manipulação de dados biológicos. Serão utilizadas apenas funções e operadores disponíveis na implementação *core* do SGBD. Caso seja necessário criar estruturas de armazenamento e/ou manipulação adicionais, estas serão definidas e apresentadas.

Conforme estipulado no Capítulo 3, as sequências biológicas serão armazenadas sob a forma de nucleotídeos (Regra 1). Além disso, toda sequência de nucleotídeo armazenada será considerada na direção 5' 3' (Regra 2). Para que a Regra 1 seja validada precisamos definir uma função que verifique se o conjunto de caracteres informado como sequência de nucleotídeos é composto apenas pelas bases A, C, G, e T. Para isto definimos a função *isDNA("sequence")*, que recebe como parâmetro uma sequência de caracteres e retorna "TRUE", caso a sequência seja formada apenas por nucleotídeos, ou "FALSE", caso contrário. A Figura 11 ilustra a implementação desta função utilizando a linguagem *plpgsql*.

Pelo fato de estarmos utilizando a estrutura de dados TEXT para representar o tipo *nucleotide sequence*, *isDNA* foi definido como uma função. Caso *nucleotide sequence* seja implementado como uma estrutura de dados nativa do SGBD, *isDNA* deverá ser implementado via *hard code* para *insert* e *update*, ou sob a forma de um *store procedure*.

```
- Name: isDNA
- Input: sequence – nucleotide sequence
- Output: boolean – true or false
- Description: verify if sequence is a DNA sequence
CREATE OR REPLACE FUNCTION isDNA(TEXT) RETURNS BOOLEAN AS
$$
   DECLARE
      original ALIAS FOR $1;
      dna BOOLEAN := TRUE;
      onechar  VARCHAR;
      length    INTEGER;
      mypos    INTEGER := 1;
   BEGIN
      SELECT UPPER(original) INTO original;
      SELECT LENGTH(original) INTO length;
      LOOP
         EXIT WHEN ((mypos > length) OR dna = FALSE);
         SELECT substring(original FROM mypos FOR 1) INTO
onechar;
          IF onechar <> 'A' AND onechar <> 'C' AND onechar
<> 'G' AND onechar <> 'T' THEN
             dna := FALSE;
          END IF;
         mypos := mypos +1;
      END LOOP;
      RETURN dna;
   END
$$
LANGUAGE plpgsql IMMUTABLE RETURNS NULL ON NULL INPUT;
```

Figura 11. Função isDNA

4.1.2.1. Função Complemento

A partir de uma sequência de nucleotídeos, a função complemento retorna uma nova sequência de nucleotídeos, que corresponde à outra fita da molécula de DNA. A construção desta função tem como base o princípio dos ligantes/pares: a base **A** sempre liga-se a base **T** e a base **G** sempre liga-se a base **C**, e vice e versa. Tendo isso como regra, devemos percorrer a sequência de entrada substituindo cada base pelo seu ligante/par. A Figura 12 apresenta esta função.

- Name: *complement*

- Input: *sequence – nucleotide sequence*

- Output: *sequence – nucleotide sequence*

- Description: *returns the complementary DNA strand*

```
CREATE OR REPLACE FUNCTION complement(TEXT) RETURNS TEXT AS
$$
   DECLARE
      original ALIAS FOR $1;
      complement TEXT := '';
      onechar  VARCHAR;
      length    INTEGER;
      mypos    INTEGER := 1;
   BEGIN
      SELECT LENGTH(original) INTO length;
      LOOP
         EXIT WHEN mypos > length;
         SELECT substring(original FROM mypos FOR 1) INTO
onechar;
          IF onechar = 'A' THEN
             onechar := 'T';
             ELSE IF onechar = 'C' THEN
                onechar := 'G';
                ELSE IF onechar = 'G' THEN
                   onechar := 'C';
                   ELSE IF onechar = 'T' THEN
                      onechar := 'A';
                    END IF;
                END IF;
             END IF;
          END IF;
         complement := complement || onechar;
         mypos := mypos +1;
      END LOOP;
      RETURN complement;
   END
$$
LANGUAGE plpgsql IMMUTABLE RETURNS NULL ON NULL INPUT;
```

Figura 12. Função complement

4.1.2.2. Função Reverso

A função reverso, como o próprio nome diz, retorna a sequencia reversa de uma sequência de nucleotídeos de entrada. Ela possibilita a realização da leitura de uma sequência de DNA na direção 3’ 5’. A ideia básica para a construção desta função é percorrer a cadeia de sequências de nucleotídeos invertendo a posição das bases. Como por exemplo, dada uma sequência de

tamanho *n*, a base localizada na posição *n* (última posição) será deslocada para a posição 1 (primeira posição). A base da posição *n-1* para a posição 2, e assim sucessivamente até a base da posição1 ser realocada para a posição *n*. A função reverso é detalhada pela Figura 13.

- Name: *reverse*

- Input: *sequence – nucleotide sequence*

- Output: *sequence – nucleotide sequence*

- Description: *returns the reverse DNA strand (reading 3' 5')*

```
CREATE OR REPLACE FUNCTION reverse(TEXT) RETURNS TEXT AS
$$
   DECLARE
      original ALIAS FOR $1;
      reversed TEXT := '\';
      onechar  VARCHAR;
      mypos    INTEGER;
   BEGIN
      SELECT LENGTH(original) INTO mypos;
      LOOP
         EXIT WHEN mypos < 1;
         SELECT substring(original FROM mypos FOR 1) INTO
onechar;
         reversed := reversed || onechar;
         mypos := mypos -1;
      END LOOP;
      RETURN reversed;
   END
$$
LANGUAGE plpgsql IMMUTABLE RETURNS NULL ON NULL INPUT;
```

Figura 13. Função reverse

4.1.2.3. Função para Obtenção do Conteúdo GC

O conhecimento sobre o conteúdo GC de uma sequência de DNA é importante para determinar propriedades físicas do DNA. Desta forma, a função de obtenção do conteúdo GC retorna a quantidade de bases G e C de uma determinada sequência de entrada.

Diferente das outras funções, que tiveram que percorrer a sequência de nucleotídeo para obter a informação desejada, a função *getGCcontent* teve sua implementação simplificada com o uso de funções predefinidas no tipo TEXT, como por exemplo a função *replace*, que substitui ocorrências de uma substring/caractere, em uma sequência, por outra substring/caractere definido.

Para a construção da função *getGCcontent*, utilizamos a função *replace* para substituir as ocorrências das bases A e T por um caractere vazio (“”), ou seja, eliminamos da sequência as bases em questão. Posteriormente, para obter o conteúdo GC da sequência, devemos apenas obter o tamanho da sequência resultante, que possui agora apenas as bases G e C. A Figura 14 ilustra a função getGCcontent.

```
- Name: getGCcontent
- Input: sequence – nucleotide sequence
- Output: integer – amount of GC content
- Description: returns the amount of GC content of DNA sequence
CREATE OR REPLACE FUNCTION getGCcontent(TEXT) RETURNS
INTEGER AS
$$
   DECLARE
      original ALIAS FOR $1;
      modify TEXT := '';
      length    INTEGER;
   BEGIN
      SELECT REPLACE(original,'A','') INTO modify;
      SELECT REPLACE(modify,'T','') INTO modify;
      SELECT LENGTH(modify) INTO length;
      RETURN length;
   END
$$
LANGUAGE plpgsql IMMUTABLE RETURNS NULL ON NULL INPUT;
```

Figura 14. Função getGCcontent.

4.1.2.4. Função Transcrição

A função que transforma uma sequência de DNA em uma sequência de RNA mensageiro é a função *transcript* (Figura 15). Sua implementação toma por base a função *getGCcontent*, necessitando de pequenas alterações. A única coisa que devemos fazer é retornar a sequência de entrada substituindo a base T pela U.

- Name: *transcript*

- Input: *sequence – nucleotide sequence (DNA)*

- Output: *sequence – nucleotide sequence (RNA)*

- Description: *returns the messenger RNA sequence*

```
CREATE OR REPLACE FUNCTION transcript(TEXT) RETURNS TEXT AS
$$
   DECLARE
      original ALIAS FOR $1;
      modify TEXT := '';
   BEGIN
      SELECT REPLACE(original,'T','U') INTO modify;
      RETURN modify;
   END
$$
LANGUAGE plpgsql IMMUTABLE RETURNS NULL ON NULL INPUT;
```

Figura 15. Função transcript

4.1.2.5. Função Tradução

Tradução é o nome dado ao processo biológico no qual a sequência de nucleotídeo de uma molécula de mRNA (RNA mensageiro) é utilizada para ordenar a síntese de uma cadeia polipeptídica com sequência de aminoácidos que determina uma proteína. Cada trinca de bases do RNA mensageiro representa um códon e está relacionada a um aminoácido específico, conforme apresentado pela Tabela 2.

Para realizar a tradução devemos percorrer a sequência de nucleotídeos de uma molécula de RNA mensageiro e, a cada sequência de três bases, converter esse trinca em um aminoácido. A conversão deve ser feita com base no sistema de tradução do código genético (Tabela 2).

Uma característica neste processo é que devemos informar/definir a posição de início de leitura das trincas, que pode ser 1, 2 ou 3. Essa informação é muito importante, pois influencia diretamente na determinação de proteínas. Por exemplo, dado a sequência de nucleotídeo “AUGCU”, se realizarmos o processo de tradução pela posição “1” teremos a trinca de bases “AUG”, que representa o aminoácido **M**et (Metiomina) que é um *start códon*. Caso a tradução comece pela posição “2” teremos a trinca de bases “UGC”, que representa o aminoácido **C**ys (Cisteína). E por fim, se começarmos a tradução pela posição

"3" teremos a trinca de bases "GCU", que representa o aminoácido **A**la (Alamina).

Como podemos ver, a função de tradução tem um requisito muito importante que é conhecer a tabela de tradução do código genético. Duas abordagens são possíveis: (1) armazenar as informações da tabela de tradução em uma estrutura auxiliar de armazenamento, ou (2) inserir o mapeamento das sequências de nucleotídeos em aminoácidos diretamente no corpo da função. Por questões de simplificação e facilidade na representação dos dados, utilizaremos a primeira alternativa, no qual os dados estarão armazenados em uma tabela. A Figura 16 representa implementação desta alternativa.

- Name: *translation*

- Input: *position – integer*

sequence – nucleotide sequence (RNA)

- Output: *sequence – amino acid sequence (Protein)*

- Description: *transforms a nucleotide sequence (RNA) in an amino acid sequence (Protein).*

```
CREATE OR REPLACE FUNCTION translation(INTEGER,TEXT) RETURNS
TEXT AS
$$
   DECLARE
      pos ALIAS FOR $1;
      seq ALIAS FOR $2;
      tam INTEGER;
      subconvert RECORD;
      sub character varying(3);
      aminoacid TEXT := '';
   BEGIN
      SELECT TRANSCRIPT(seq) INTO seq;
      SELECT LENGTH(seq) INTO tam;
      LOOP
         EXIT WHEN pos+2 > tam;
         SELECT substring(seq FROM pos FOR 3) INTO sub;
         SELECT INTO subconvert * FROM code WHERE triplet =
sub;
         aminoacid := aminoacid || subconvert.aa;
         pos := pos +3;
      END LOOP;
      RETURN aminoacid;
   END
$$
LANGUAGE plpgsql IMMUTABLE RETURNS NULL ON NULL INPUT;
```

Figura 16. Função translation

Conforme já mencionado, a tradução é realizada sobre uma sequência de RNA. Para evitar erros por parte do usuário que por desconhecer esta característica biológica possa informar uma sequência de DNA ao invés de RNA, a função realiza a transcrição da sequência de entrada antes de efetivamente realizar a tradução.

4.1.2.6. Função Procura ORF

A região transcrita do DNA, que gera uma proteína qualquer, é chamada de ORF (*Open Read Frame*). A principal característica de uma ORF é que ela é uma subsequência de DNA transcrito delimitada pelos códons de iniciação (AUG) e terminação (UAA, UGA e UAG).

Da mesma forma que a função de tradução, devemos informar/definir a posição de início de leitura das trincas, que pode ser 1, 2 ou 3. Essa informação influencia diretamente na identificação dos códons de iniciação e terminação. Adicionalmente, devemos informar um tamanho mínimo que a ORF deve ter.

Para realizar a busca por ORFs devemos percorrer a sequência de nucleotídeos de uma molécula de RNA mensageiro, encontrar o *start códon* e, a partir do tamanho mínimo, o *stop códon*.

Da mesma forma que a tradução, a procura de ORFs é realizada sobre uma sequência de RNA. Para evitar erros por parte do usuário que por desconhecer esta característica biológica possa informar uma sequência de DNA ao invés de RNA, a função realiza a transcrição da sequência de entrada antes de efetivamente realizar a procura. A Figura 17 ilustra esta função.

- Name: *searchORF*

- Input: *position – integer*

sequence – nucleotide sequence (RNA)

length - minimum size of ORF

- Output: *sequence Collection – amino acid sequence (Protein)*

- Description: *search ORFs in a nucleotide sequence (RNA).*

```
CREATE OR REPLACE FUNCTION searchORF(INTEGER,TEXT,INTEGER)
RETURNS SETOF TEXT AS
$$
   DECLARE
      pos ALIAS FOR $1;
      seq ALIAS FOR $2;
      size ALIAS FOR $3;
      tam INTEGER;
      tamORF INTEGER;
      sub character varying(3);
      orf TEXT := '';
      found BOOLEAN := false;
   BEGIN
      SELECT TRANSCRIPT(seq) INTO seq;
      SELECT LENGTH(seq) INTO tam;
      LOOP
         EXIT WHEN pos+2 > tam;
         SELECT substring(seq FROM pos FOR 3) INTO sub;
         IF (sub = 'AUG') THEN
            found := true;
         END IF;
         IF (found) THEN
            orf := orf || sub;
         END IF;
         IF (sub = 'UAA' OR sub = 'UGA' OR sub = 'UAG') THEN
            found := false;
            SELECT LENGTH(orf) INTO tamORF;
            IF (tamORF >10) THEN
               RETURN QUERY SELECT orf;
            END IF;
            orf := '';
         END IF;
         pos := pos +3;
      END LOOP;
   END
$$
LANGUAGE plpgsql IMMUTABLE RETURNS NULL ON NULL INPUT;
```

Figura 17. Função searchORF

4.2. Modelo de Dados Biológicos

O modelo conceitual de dados biológico proposto no Capítulo 3 foi utilizado como referência para representar, armazenar e disponibilizar dados biológicos do projeto *Protein World DB* (PWD). A utilização do modelo neste projeto foi de extrema importância, pois foi possível verificar a representatividade do esquema utilizando um caso real para teste.

Além da questão de representação, armazenamento e manipulação de sequências biológicas pode-se efidenciar outras questões que trazem certa complexidade ao domínio, que são as dificuldades de representação, armazenamento e manipulação de outros dados em banco de dados relacional, e.g. HITs, recursão e hierarquias.

O restante deste Capítulo dedica-se à apresentação da geração do esquema lógico e descrição de questões levantadas durante a pesquisa deste projeto e das soluções propostas e implementadas, que dizem respeito à implementação do esquema e experimentação através de um conjunto de consultas ditas básicas e complexas.

4.2.1. Extensão do Modelo Conceitual

Adicionalmente, foram incorporadas ao modelo conceitual informações relacionadas a anotações de proteínas. De acordo com [NCBI Annotation Information 2012], anotação é processo de atribuição das funções biológicas preditas e características estruturais aos dados brutos, e.g. à sequência primária da proteína. Vale ressaltar que a predição das funções celulares (estruturais, enzimas, transportadores, sinalizadores, etc.) é em sua maioria hipotética. A maior parte dessas possíveis funções foi atribuída por análises *in silico* e somente uma pequena fração das proteínas preditas teve suas funções confirmadas por experimentos laboratoriais.

Para o uso neste projeto, o modelo conceitual proposto foi estendido com a adição das seguintes informações de anotação:

- Enzimas/Vias metabólicas (KEGG): são um grupo de substâncias orgânicas de natureza, normalmente proteica (existem também enzimas constituídas de RNA, as ribozimas), com atividade intracelular ou extracelular, as quais possuem funções catalisadoras [Kanehisa et. al. 2012] [Kanehisa and Goto 2000];

- Domínio (Pfam): Pfam é um banco de família e domínio de proteínas, os quais são representados por uma coleção de alinhamentos múltiplos de sequências e modelos de Markov [Punta et. al. 2012] [Finn et. al. 2010];
- Ontologia (Gene Ontology): Gene Ontology (GO) é uma ontologia de domínio, formada por três categorias de conceitos, a saber: Função Molecular, Processo Biológico e Componente Celular. Ela tem por objetivo produzir um vocabulário controlado que possa ser aplicado a todos os organismos, para representação do conhecimento na descrição de genes e papéis de proteínas em células [The Gene Ontology Consortium 2010] [The Gene Ontology Consortium 2008].

  A Figura 18 ilustra o relacionamento entres as anotações e proteínas.

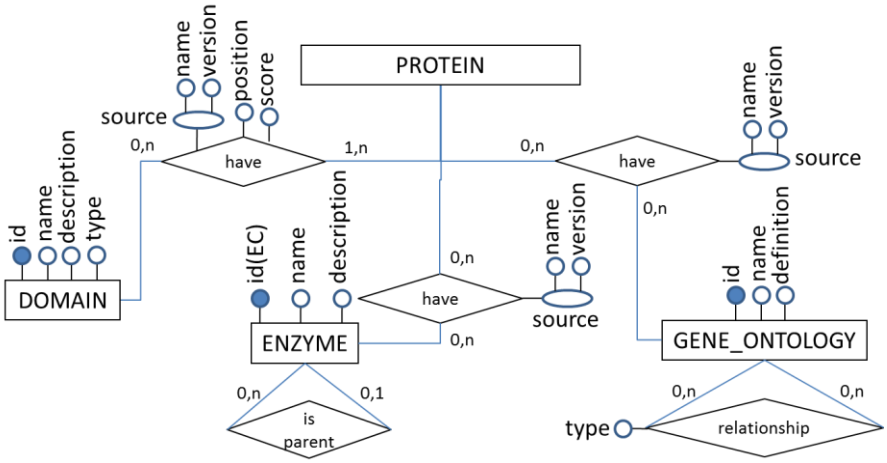

Figura 18. Diagrama conceitual estendido: anotações e proteínas

4.2.2. Modelo Lógico

Uma vez definido o esquema conceitual, conforme descrito no Capítulo anterior, podemos realizar o mapeamento e a geração do esquema lógico. Normalmente surgem dúvidas quanto à diferença entre modelo conceitual e lógico. Resumidamente, a modelagem conceitual é o mais alto nível de abstração, onde busca-se identificar todas as entidades e os relacionamentos existentes entre elas, sem limitações ou aplicação de tecnologia específica. O diagrama de Entidade e Relacionamento (ER) é o mais usado. O esquema lógico, por sua vez, já leva em conta algumas limitações e implementa alguns recursos como adequação de padrão e nomenclatura. Também se deve pensar no conceito de chaves primárias e estrangeiras, levando em conta as entidades e relacionamento criados no esquema conceitual. É uma preparação para o esquema físico (esquema gerado para um SGBD específico), que contempla também desempenho.

No processo de transição do esquema conceitual para o esquema lógico, foram utilizadas algumas regras básicas de mapeamento e, quando necessário, algumas decisões de modelagem foram tomadas. De acordo com uma dessas regras, todas as entidades presentes no esquema conceitual tornam-se tabelas no esquema lógico com seus respectivos atributos. Deste modo, as entidades PROTEIN, ORF_T, GENOMIC SEQUENCE, CDS, GENE, TAXONOMY, RANK, DOMAIN, GENE_ONTOLOGY e ENZYME foram mapeadas para as tabelas protein, orf_t, genomicsequence, cds, gene, taxonomy, tax_rank, domain, gene_ontology e enzyme respectivamente.

Levando em consideração a aplicação dessa regra, o restante deste capítulo apresenta o mapeamento dos relacionamentos presentes no esquema conceitual para o lógico, descrevendo as outras regras empregadas, e algumas das particularidades e dificuldades encontradas durante o processo. Por fim, apresentamos o resultado desta transformação como um todo, e detalhamos os atributos de cada tabela resultante.

4.2.2.1. Hits

O resultado da primeira etapa do Projeto de Comparação de Genomas foi o conjunto de arquivos contendo dados de similaridade (hits) entre proteínas e ORFs traduzidas (tORFs). Como a relação de similaridade entre proteínas e tORFs é do tipo “muitos-para-muitos”, foi preciso criar uma tabela intermediária para cada relação, tendo como chave primária a composição das chaves primárias das outras duas tabelas envolvidas no relacionamento. As tabelas criadas foram: (a) hits_oo, relação entre tORFs, (b) hits_pp, relação entre proteínas, e (c) hits_op, relação entre tORFs e proteínas. A Figura 19 ilustra esse processo de mapeamento.

As tabelas intermediárias (hits) devem receber os atributos que identificam a relação entre as entidades. Neste caso, cada tabela possui os atributos que identificam o alinhamento entre cada proteína/tORF, que são: query_fiocruzid (identificador da sequência query), subject_fiocruzid (identificador da sequência subject); sw_score (pontuação Smith-Waterman), bit_score, e_value, identity (percentual de identidade), alignment, (tamanho do alinhamento), query_start (início da sequência query), query_end (final da sequência query), subject_start (início da sequência subject), subject_end (final da sequência subject), query_gaps (gaps na sequência query), subject_gaps (gaps na sequência subject).

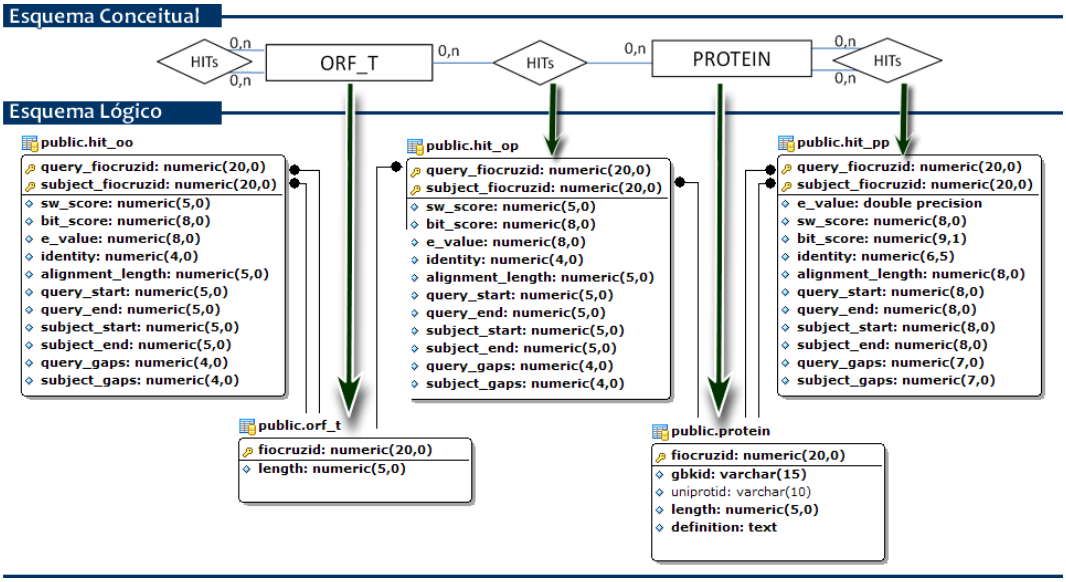

Figura 19. Mapeamento da similaridade (hits) entre proteínas e tORFs

4.2.2.2. Taxonomia

De acordo com [The NCBI Handbook 2012], taxonomia é uma excelente forma de organização no estudo de sistemas biológicos. Herança, homologia por descendência comum e à conservação de sequência e estrutura na determinação da função são todas as ideias centrais em biologia que estão diretamente relacionados com a história evolutiva de um grupo de organismos. Para o mapeamento da taxonomia, os relacionamentos devem ser analisados em separado. A Figura 20 ilustra o resultado da aplicação das regras de transformação explicitadas abaixo:

- Relacionamento entre taxonomias: é um tipo de auto relacionamento “um-para-muitos”. Com ele representamos a linhagem taxonômica dos indivíduos, onde cada nodo apresenta um link (referência) para o seu nível taxonômico superior. Dessa forma a tabela *taxonomy* recebe um novo atributo denominado “*father*” (*foreing key* para *taxonomy*). A cardinalidade “0.1” da extremidade filha indica que essa referência pode ser nula, caso usado na representação de nodos do tipo raiz, os quais não possuem nodo pai.
- Relacionamento entre taxonomia e rank: esse é um tipo de relacionamento “um-para-muitos”. Por isso a tabela taxonomia recebe o atributo de referência ao rank relacionado, denominado tax_rank_id. O

fato da cardinalidade ser “1.1” indica que a taxonomia deve possuir um e somente um rank, ou seja, não aceita valor nulo.

- Relacionamento entre taxonomia e proteína/tORF: da mesma forma que o anterior, esse é um tipo de relacionamento “um-para-muitos” de cardinalidade “1.n”. Por isso tanto a tabela protein quanto a tabela orf_t recebem o atributo de referência taxonomy_id. Isso indica que proteínas e torfs devem possuir uma taxonomia, e por sua vez, uma linhagem taxonômica.

Uma característica muito importante é que o atributo *synonymous* é do tipo composto. Por isso houve a necessidade de retirar esse atributo da tabela *taxonomy* e de criar uma tabela denominada “*synonymous*” que possui os diferentes nomes de um mesmo indivíduo. Ela também deve possuir o taxonomy_id (*foreing key*) para referenciar o indivíduo relacionado.

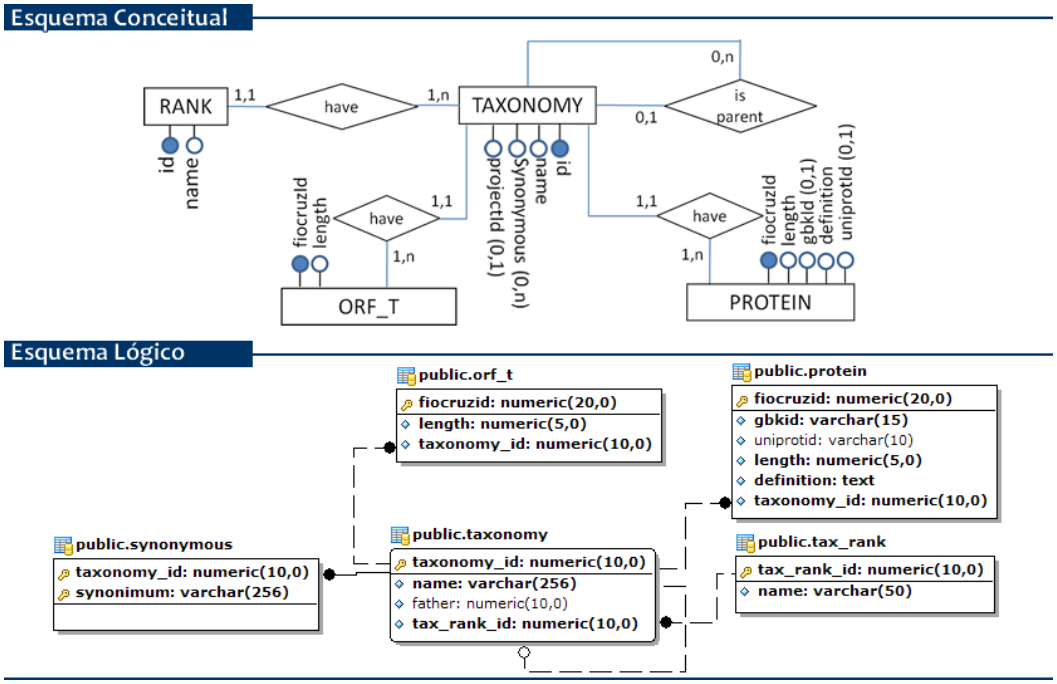

Figura 20. Mapeamento da taxonomia

4.2.2.3. Dogma Central da Biologia Molecular

A seguir são apresentadas as regras de transformação utilizadas no mapeamento dos relacionamentos envolvidos na parte destinada ao dogma central da biologia molecular, conforme ilustrado pela Figura 21:

- Relacionamento entre tORFs e sequência genômica: de acordo com o esquema conceitual, uma sequência genômica pode conter zero ou mais orfs, e uma orf é proveniente de uma e somente uma sequência genômica. Desta forma, esse é um tipo de relacionamento “um-para-

muitos". Uma alternativa para de mapeamento para o modelo lógico seria introduzir na tabela torf uma referência para a sequência genômica que a origina, além dos atributos existentes nesse relacionamento. Contudo, foi decidido criar uma tabela denominada de orf_region para armazenar os atributos deste relacionamento por motivos organizacionais e pelo fato de representarem a região da sequência genômica onde a orf é obtida. Por fim, foram acrescentados na tabela orf_region os atributos de referência à torf, fiocruzid (primary key), e sequência genômica, gbkid (foreing key).

- Relacionamentos de CDS: como CDS possui uma relação do tipo "um-para-muitos" com sequência genômica e gene, cada relação é convertida na inserção de atributos de referência, gbkid e geneid respectivamente, à "tabela" CDS. Por sua vez, CDS possui um relacionamento do tipo "um-para-um" com proteína. Para este tipo de relacionamento existe mais de uma forma de mapeamento. No entanto, optou-se por referenciar a tabela proteína em CDS pelo atributo fiocruzid.

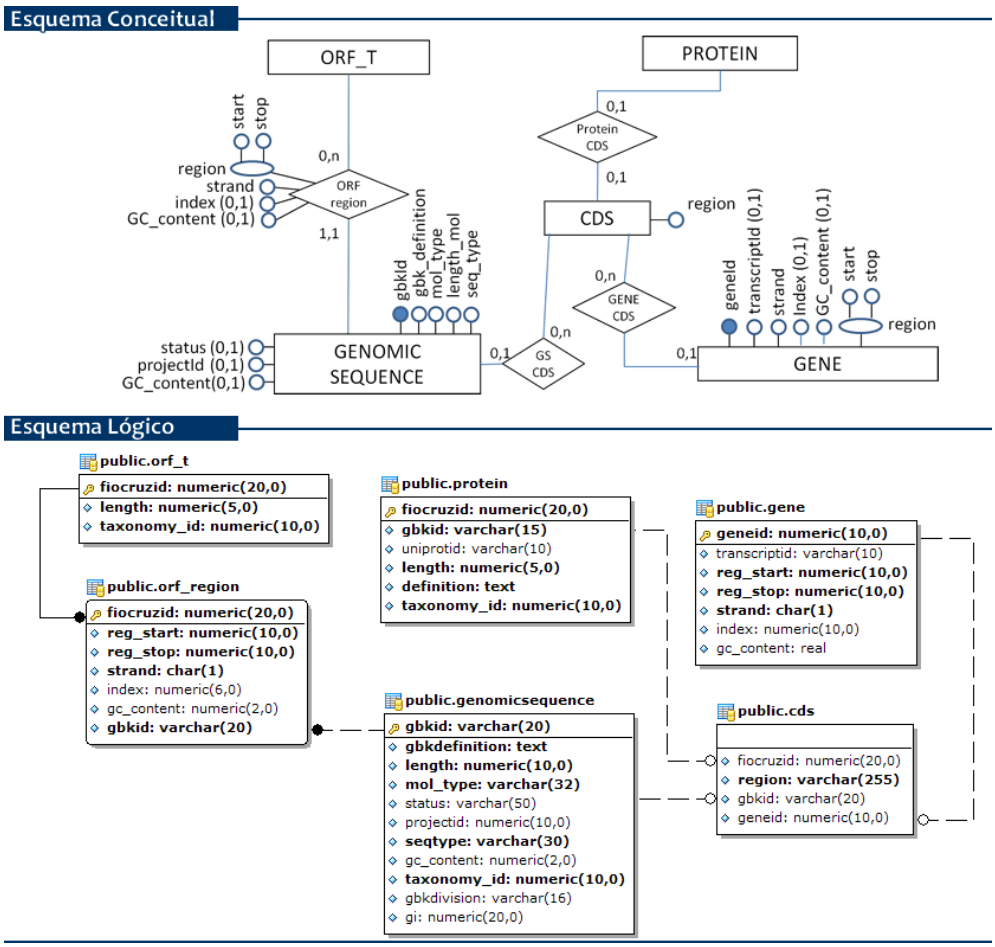

Figura 21. Mapeamento do dogma central

Neste ponto do mapeamento houve uma segunda mudança no esquema lógico em relação ao conceitual. Para facilitar o processo de busca e melhorar o desempenho, foi acrescentado na tabela *genomicsequence* uma referência para taxonomia (taxonomy_id). Assim, essa informação, que é muito utilizada em consultas referentes à sequência genômica, é obtida diretamente sem a necessidade de consultar (*join*) outras tabelas.

4.2.2.4. Anotações

Com base nas informações apresentadas na Seção 4.2.1, conclui-se que o relacionamento de proteínas com suas anotações é do tipo “muitos-para-muitos”. Sendo assim, o mapeamento é realizado criando-se uma tabela intermediária para cada relacionamento, os quais terão como chave primária a composição das chaves primárias das outras duas entidades/tabelas envolvidas, podendo ainda acrescentar os atributos que identificam essa relação.

Como resultado do mapeamento da relação da entidade *protein* entre as entidades *domain*, *enzyme* e *gene ontology*, foram criadas as tabelas *domainannotation*, *enzyneannotation* e *goannotation* respectivamente. Por fim, foram mapeados os auto relacionamentos de *enzyme* e *gene ontology*, os quais podem ser visualizados graficamente pela Figura 22:

- Enzyme: por ser um auto relacionamento do tipo “um-para-muitos” foi aplicada a mesma regra empregada em taxonomia. A tabela *enzyme* recebe um novo atributo denominado “*father*” (*foreing key* para enzyme), podendo ser nulo, conforme cardinalidade presente no esquema conceitual (“0.1”).
- Gene ontology: o auto relacionamento de *gene ontology* é do tipo “muitos-para-muitos”. Por esse motivo foi criada a tabela *relationship*. Nela foram acrescentadas as referências para a tabela *ontology*, o atributo tipo de relacionamento (relationship_type) e um atributo sequencial denominado relationship_id (*primary key*). A necessidade da criação desse atributo para identificação de chave primária, e a não utilização dos atributos de referência à ontologia como chave, deve-se ao fato da existência de diferentes tipos de relacionamentos para uma mesma relação.

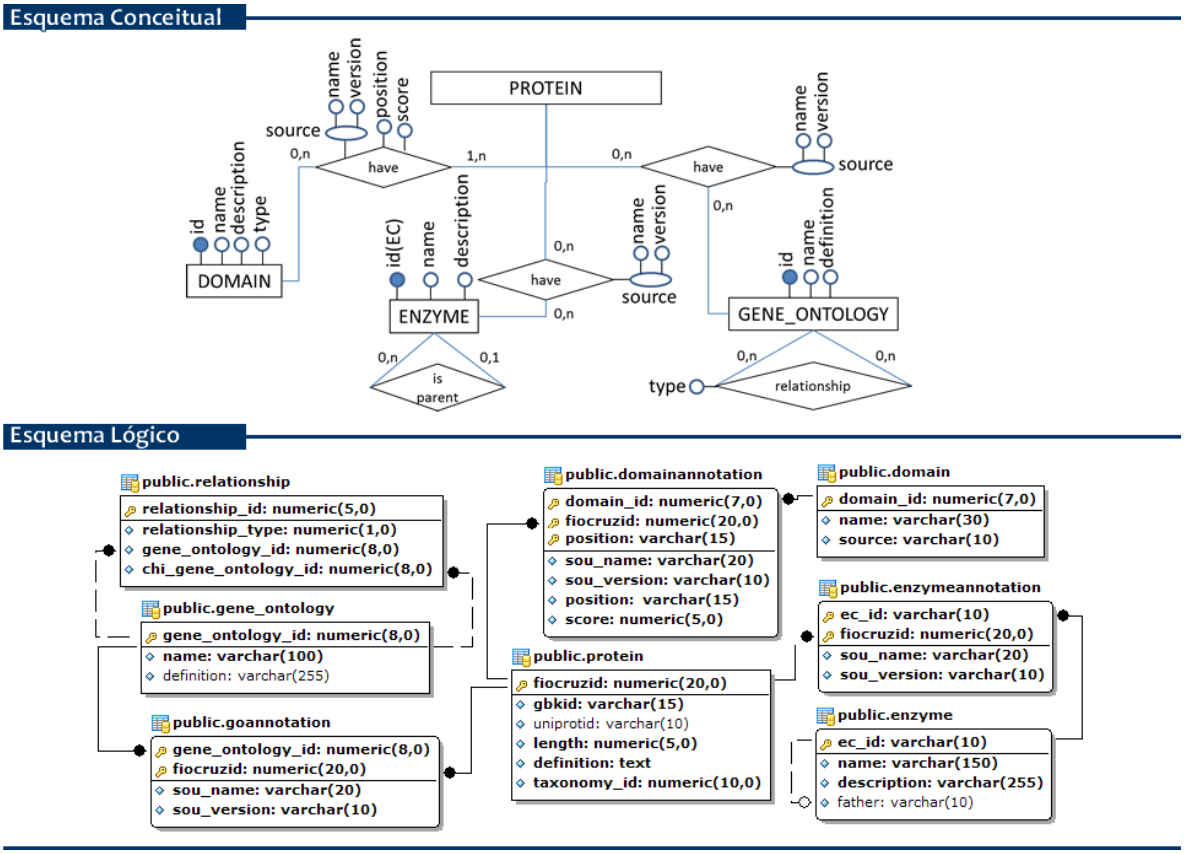

Figura 22. Mapeamento das anotações

4.2.2.5. Visão Geral

A Figura 23 ilustra o esquema lógico resultante do mapeamento, sobre o modelo conceitual (Figura 10), utilizando o conjunto de regras de transformação descrito nesse capítulo.

Outra característica muito importante na elaboração do esquema lógico está relacionada ao tipo e tamanho definidos para cada atributo. Pesquisas na literatura e, principalmente a expertise do grupo de pesquisa, foram fundamentais na concepção do esquema. A seguir são descritos os principais atributos que necessitaram de uma pesquisa mais profunda para sua definição:

- Identificador de proteína/tORF (fiocruzid): por não haver uma padronização, cada projeto de pesquisa ou banco de dados público define e utiliza um identificador próprio para proteína. Para o Projeto Comparação de Genoma não foi diferente. O processo de comparação envolveu proteínas de diferentes bases, e consequentemente diferentes identificadores, e sequências de aminoácidos não catalogadas, as tORFS, que receberam um identificador próprio. Todos esses identificadores são numéricos e nenhum ultrapassa a casa das 1020

posições. Sendo assim, nosso identificador de proteínas/tORFs, e demais referências, foram definidos como numérico de 20 posições.

- Atributos de tamanho e posição: todos os atributos que se referem ao tamanho ou posição em sequências de aminoácidos foram definidos com tamanho 5, pois de acordo com estatísticas do release 57.8 do UniProtKB/Swiss-Prot, datado em 22/09/2009, a maior sequência de aminoácidos presente na base é a da proteína TITIN_MOUSE (A2ASS6) que possui a quantidade de 35213 aa (aminoácidos), enquanto que atributos que se referem ao tamanho ou posição em sequências de nucleotídeos foram definidos com tamanho 10, pois, de acordo com dados do Projeto Genoma Humano (HGP), coordenado pelo *U.S. Department of Energy and the National Institutes of Health*, o comprimento do cromossoma humano varia de 51 milhões a 245 milhões de bp (base-par), e o gene dystrophin é o maior gene encontrado na natureza medindo 2.4 Mb, segundo dados do Entrez Gene NCBI [Entrez Gene 2010].

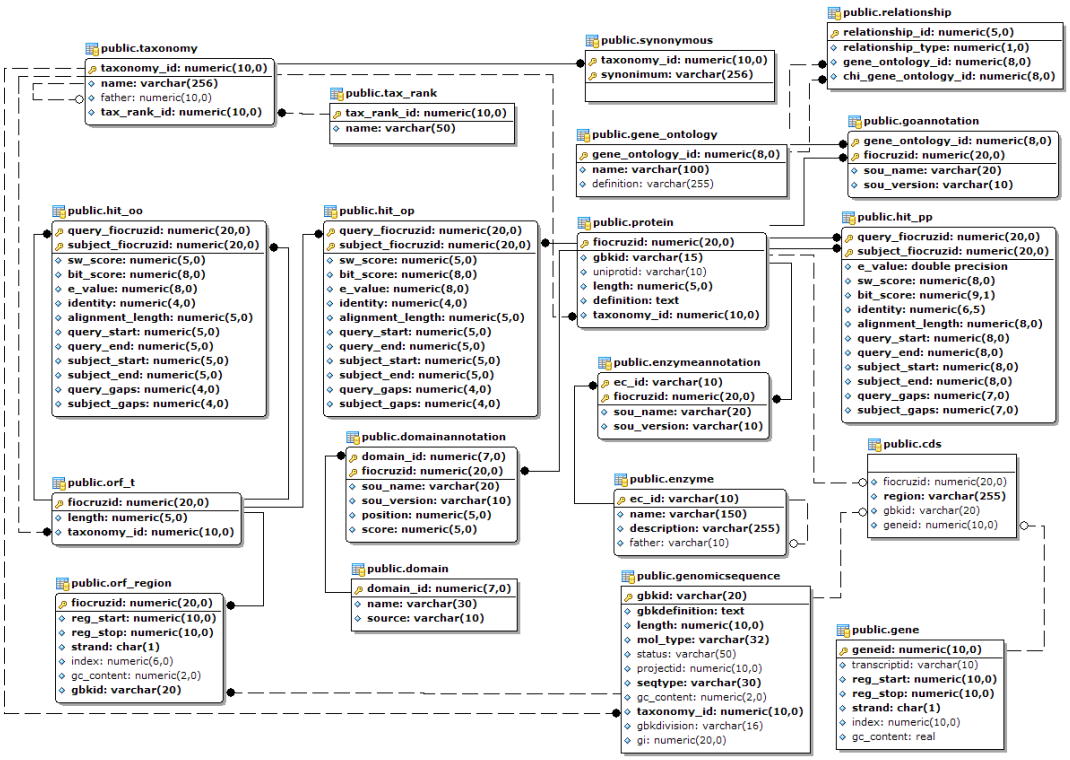

Figura 23. *Protein World Database*: esquema lógico

4.2.3. Definição de Consultas

Mais do que representar e relacionar os conceitos envolvidos no domínio, o esquema deve ser capaz de responder questões conceituais e teóricas sobre

os objetos representados. A seguir serão apresentadas algumas questões de análise sobre o modelo conceitual descrito na Seção 3.2.1, divididas em básicas e complexas.

Para verificar se o esquema realmente consegue fornecer a informação às questões sugeridas, consultas foram implementadas utilizando como estudo de caso o banco de dados do PWD. O banco de dados PWD utiliza o SGBD relacional PostgreSQL, versão 9.1, e o esquema implementado é referente ao modelo apresentado na Seção 3.2.1. Vale salientar que sem a utilização da solução proposta nem mesmo as consultas "básicas" seriam respondidas facilmente.

4.2.3.1. Questões Básicas de Análise

As questões ditas "básicas" são questões de análise onde a informação diz respeito apenas a dados estatísticos de um ou mais objetos como, por exemplo, contabilizações de ocorrências/registros em um determinado objeto, ou entre objetos que possuem relacionamento entre si. Este tipo de análise pode parecer simples mas, se observado sob o ponto de vista biológico, são questões muito importantes que fornecem valiosas informações.

4.2.3.1.1. Quantidade de Proteínas Comparadas

Obter a quantidade de proteínas comparadas, conceitualmente, não faz sentido, pois todas as proteínas possuem ou não similaridade (*hits*) com outras proteínas. Como a informação de similaridade é obtida através de um processo externo, podemos ter dois tipos distintos de implementação.

Uma alternativa é representar apenas proteínas que tenham passado pelo processo de alinhamento. Desta forma, descobrir a quantidade de proteínas comparadas seria simplesmente contar o número de proteínas (objeto *Protein*). A segunda alternativa, abordagem esta implementada neste projeto, seria representar o universo de todas as proteínas conhecidas (comparadas ou não). Para este tipo de implementação, devemos acrescentar uma "*flag*" para identificar aquelas que participaram do processo de alinhamento. Proteínas com *flag* igual a 1 (positivo) identifica que esta participou do processo de comparação.

Sabendo desta informação, a consulta que representa a quantidade de proteínas comparadas no PWD é a representada pela Figura 24.

```
SELECT COUNT (fiocruzid)
FROM PROTEIN
WHERE compared = b'1’;
```

Figura 24. Consulta: Quantidade de Proteínas Comparadas

4.2.3.1.2. Proteínas com Sequência Genômica de Origem

Da mesma forma que a consulta anterior, conceitualmente essa questão não faz sentido, pois toda proteína é originada de uma sequência genômica. Contudo, na prática, podemos ter proteínas onde não conhecemos sua sequência genômica de origem, pois tal proteína foi originada de um processo automático de identificação ou através de manipulação em bancada.

Outra informação conceitual importante é que toda proteínas é originada de uma sequência genômica a partir de um CDS (*coding sequence*), que é a região codificadora da proteína em uma sequência genômica de um gene. Deste modo, para responder esta questão, devemos verificar quantas proteínas (entidade *Protein*) possuem relação com uma região codificadora de sequência genômica (entidade *CDS*). A consulta que representa a quantidade de proteínas com sequência genômica de origem é a representada pela Figura 25.

```
SELECT COUNT (CDS.fiocruzid)
FROM CDS
WHERE CDS.fiocruzid is not null;
```

Figura 25. Consulta: Proteínas com Sequência Genômica de Origem

4.2.3.1.3. Quantidade de Genomas que Pertencem a um Grupo Taxonômico

Em primeiro lugar, para obtermos a quantidade de genomas que pertencem a um grupo taxonômico, devemos saber qual o identificador para o determinado grupo taxonômico que se deseja pesquisar. Após, devemos obter todos os identificadores de táxons (*taxonomy_id*) abaixo do grupo taxonômico escolhido. Obter este tipo de informação requer um processo de “pesquisa em árvore”. No SGBD PostgreSQL podemos realizar esta consulta utilizando a ideia de consulta recursiva com o auxilio do comando “WITH” e *Common Table Expressions* (CTE), conforme ilustrado pela Figura 26.

```
WITH RECURSIVE path(id, parent_id) AS (
      SELECT taxonomy_id, father
      FROM taxonomy t
      WHERE taxonomy_id = valor //definer valor
   UNION
      SELECT t.taxonomy_id, t.father
      FROM taxonomy t, path as parentpath
      WHERE t.father = parentpath.id)
SELECT id FROM path;
```

Figura 26. Consulta: Obtenção de Nodos de uma Árvore Taxonômica.

Definida a declaração para obtenção dos nodos filhos de uma árvore, podemos criar uma função para facilitar o processo de procura dos descendentes de qualquer nó da árvore (Figura 27).

- Name: *getTaxonomyIdChildren*

- Input: *integer - taxonomy_Id*

- Output: *numeric collection - taxonomy_Id*

- Description: *get taxonomy_Id children of a specific taxonomy_Id.*

```
CREATE OR REPLACE FUNCTION getTaxonomyIdChildren(INTEGER)
RETURNS SETOF NUMERIC AS
$$
WITH RECURSIVE path(id, parent_id) AS (
   SELECT taxonomy_id, father
   FROM taxonomy t
   WHERE taxonomy_id = $1
      UNION
   SELECT t.taxonomy_id, t.father
   FROM taxonomy t, path as parentpath
   WHERE t.father = parentpath.id)
SELECT id FROM path;
$$
LANGUAGE "sql" IMMUTABLE;
```

Figura 27. Função getTaxonomyIdChildren.

Agora, por questões de diferenças e incompatibilidades entre linguagens, o tipo de saída da função SQL da Figura 27 é incompatível com os tipos de dados manipulados por uma função PLPGSQL. Desta forma, uma nova consulta teve quer ser implementada. Ela serve apenas como um conversor de tipo para

ser reusada por outras funções PLPGSQL. Esta função é ilustrada pela Figura 28.

- Name: *getTaxonomyIdChildrenSet*

- Input: *integer - taxonomy_Id*

- Output: *numeric collection - taxonomy_Id*

- Description: *get taxonomy_Id children of a specific taxonomy_Id.*

```
CREATE OR REPLACE FUNCTION getTaxonomyIdChildrenSet(INTEGER)
RETURNS SETOF NUMERIC AS
$$
   BEGIN
      RETURN QUERY select * from gettaxonomyidchildren($1);
   END
$$
LANGUAGE plpgsql IMMUTABLE RETURNS NULL ON NULL INPUT;
```

Figura 28. Função getTaxonomyIdChildrenSet.

O passo seguinte do processo é contar a quantidade de espécies pertencentes a este grupo que tem sequências genômica (origem das proteínas comparadas) na entidade *Genomic Sequence*. O resultado é obtido comparando o taxonomy_id das sequencias genômicas com o taxonomy_id das espécies pertencentes ao grupo taxonômico desejado. Para facilitar o uso da pesquisa, a consulta que retorna a quantidade de genomas pertencentes a um grupo taxonômico será apresentada já na forma de função (Figura 29).

Analisando a função que retorna a quantidade de genomas de uma determinada taxonomia, percebemos que a consulta realiza um filtro utilizando o gbkid (*genbank identification*) da sequência genômica. Isto porque os genomas são representados por:

- AC_
    - Este prefixo é utilizado para moléculas genômicas que são fornecidos para refletir uma anotação ou *assembly* alternativo. Principalmente usado para registros procarióticos e vírus.
- NC_
    - Utilizado para representar moléculas genômicas completes, incluindo genomas, cromossomas, organelas, plasmídeos.

- NG_
    - Região genômica incompleta; fornecido para suportar o pipeline de anotação de genoma do NCBI. Representa pseudogenes não transcritos, ou regiões de maior dimensão que representam um agrupamento de genes que é difícil para anotar através de métodos automáticos.
- NT_
    - Assembles de genomas intermediários de BAC e/ou genoma completo.
- NW_
    - Assembles de genomas intermediários de BAC ou genoma completo.

- Name: *getCountGenomeTaxonomy*

- Input: *integer - taxonomy_Id*

- Output: *bigint - amount of gbkid*

- Description: *get the amount of gbkid belonging to a specific taxonomy_Id.*

```
CREATE OR REPLACE FUNCTION getCountGenomeTaxonomy (INTEGER)
RETURNS BIGINT AS
$$
SELECT count(*) FROM (
   SELECT (gs.gbkid) FROM genomic_sequence gs
   WHERE gbkid like 'AC_%' and gs.taxonomy_id IN
     (select * from gettaxonomyidchildrenset($1))
UNION
   SELECT (gs.gbkid) FROM genomic_sequence gs
   WHERE gbkid like 'NC_%' and gs.taxonomy_id IN
     (select * from gettaxonomyidchildrenset($1))
) as t
$$
LANGUAGE "sql" IMMUTABLE;
```

Figura 29. Função getCountGenomeTaxonomy.

- NZ_
    - Uma coleção de dados de sequências de genomas completos inseridas para um projeto. As adesões não são rastreadas entre *releases*. Os quarto primeiros caracteres que seguem após o *underscore* (e.g. 'ABCD') identificam o projeto.

- NS_
    - Registros de genomas que representam um assemble que não reflete a estrutura de uma molécula biológica real.

Deste modo, estamos interessando apenas nas sequências genômicas que possuem os identificadores “AC_” e “NC_”.

4.2.3.1.4. Quantidade de Proteínas que Pertencem a um Grupo Taxonômico

Esta consulta é similar à anterior (Seção 4.2.3.1.3). A diferença é que devemos contar todas as proteínas originadas dos genomas pertencentes a um determinado grupo taxonômico. No entanto, este processo é facilitado, uma vez que nosso modelo traz algumas abstrações.

Conforme já mencionado, o modelo proposto tenta ao máximo ser fiel aos conceitos utilizados na teoria da biologia molecular. Contudo, para satisfazer algumas necessidades e realidades enfrentadas na prática, o modelo sofreu algumas abstrações. Um exemplo é que na teoria toda proteína é originada de um genoma, mas na prática, muitas vezes as proteínas são sequenciadas sem conhecer o genoma de origem. Deste modo o objeto/entidade *Protein* possui como atributo a informação do *taxonomy_Id*, pois mesmo desconhecendo o genoma, a espécie sequenciada é conhecida.

A função que retorna a quantidade de proteínas que pertencem a um grupo taxonômico é apresentada abaixo pela Figura 30.

- Name: *getCountProteinTaxonomy*

- Input: *integer - taxonomy_Id*

- Output: *bigint - amount of fiocruzid*

- Description: *get the amount of fiocruzid belonging to a specific taxonomy_Id.*

```
CREATE OR REPLACE FUNCTION getCountProteinTaxonomy (INTEGER)
RETURNS BIGINT AS
$$
SELECT COUNT(p.fiocruzid) FROM protein p
WHERE p.taxonomy_id IN (select * from
gettaxonomyidchildrenset($1));
$$
LANGUAGE "sql" IMMUTABLE;
```

Figura 30. Função getCountProteinTaxonomy.

4.2.3.1.5. Quantidade de Hits para uma Proteína “X”, dado uma Linha de Corte

Este tipo de informação é muito útil para a análise e descoberta de novas proteínas. A quantidade de hits que uma proteína obteve no processo de alinhamento é obtida através do número de ocorrências desta proteína na tabela/relacionamento *hit_pp*. A ocorrência pode ser tanto em *query_fiocruzid* quanto em *subject_fiocruzid*. Além disso, só devem ser considerados os hits que estiverem abaixo de um determinado *e-value*. A seguir esta consulta é apresentada na forma de função (Figura 31).

Com pequenas modificações nesta função podemos obter uma gama maior de informação. Caso seja de interesse do usuário, poderíamos verificar o número de hits de uma proteína levando em consideração somente as ORFs ou proteínas e ORFs. No primeiro caso ao invés de realizar a busca na tabela/relação *hit_pp* utilizaríamos a tabela/relação *hit_op* e compararíamos apenas o *subject_fiocruzid*, no qual representa a proteína. No segundo caso deveríamos somar a busca realizada em *hit_pp* e *hit_op*.

- Name: *getCountHitsProtein*

- Input: *numeric - taxonomy_Id*

numeric - evalue

- Output: *integer - amount of hits*

- Description: *gets the amount of hits in a protein given an e-value.*

```
CREATE OR REPLACE FUNCTION getCountHitsProtein (NUMERIC,
DOUBLE PRECISION) RETURNS BIGINT AS
$$
SELECT COUNT (*) FROM (
   SELECT (hq.query_fiocruzid) FROM hit_pp hq
   WHERE hq.query_fiocruzid = $1 AND hq.e_value <= $2
   UNION ALL
   SELECT (hq.subject_fiocruzid) FROM hit_pp hq
   WHERE hq.subject_fiocruzid = $1 AND hq.e_value <= $2
) as t;
$$
LANGUAGE "sql" IMMUTABLE;
```

Figura 31. Função getCountHitsProtein.

Outra forma de estender esta consulta seria utilizar como critério de filtro, não apenas *o e-value*, mas sim, todos os atributos resultantes do processo de

comparação, existentes nas tabelas de hits: *SW score* (pontuação Smith-Waterman), *bit score*, *% identity* (percentual de identidade), *alignment length* (tamanho do alinhamento), *query start* (início da sequência *query*), *query end* (final da sequência *query*), *subject start* (início da sequência *subject*), *subject end* (final da sequência *subject*), *query gaps* (gaps na sequência *query*), *subject gaps* (gaps na sequência *subject*).

4.2.3.2. Consultas Complexas

Diferentemente das consultas simples, que buscam apenas sumarizações entre objetos, as consultas complexas envolvem conceitos e conhecimento mais aprofundados da biologia molecular. Levando em consideração o modelo conceitual da Figura 10, que foi estruturado com base nos conceitos da biologia molecular, o processo de obtenção deste tipo de informação é facilitado. A seguir, são apresentadas consultas para obtenção de informações biológicas, tais como a descoberta de genes únicos e genes homólogos (ortólogos e parálogos).

4.2.3.2.1. Genes Únicos

Genes únicos são genes encontrados exclusivamente em certos grupos de organismos. Diferentes grupos de organismos têm genes diferentes que não se encontram em outros grupos. A descoberta de genes únicos é muito importante para a pesquisa de doenças e características específicas de indivíduos, e.g. os “traços mendelianos”. O termo “traço mendeliano” é usado para descrever uma característica causada por um único gene que por vezes reaparece num quarto da descendência. Essa característica pode ser inócua, tal como as sardas ou a capacidade de dobrar a língua, mas também pode conduzir a doenças graves.

No processo de busca por genes únicos, devemos encontrar as proteínas, a partir de um proteoma completo de uma dada espécie, que não são similares a qualquer proteína de outro proteoma de uma espécie diferente. Por exemplo, no nível de gênero devemos identificar as proteínas pertencentes a proteomas completos de diferentes espécies em relação a este gênero, podendo ser semelhantes dentro do grupo taxonômico, mas sem similaridade com as proteínas fora do gênero correspondente.

A Figura 32 ilustra, utilizando a teoria dos conjuntos, como deve ser o resultado esperado.

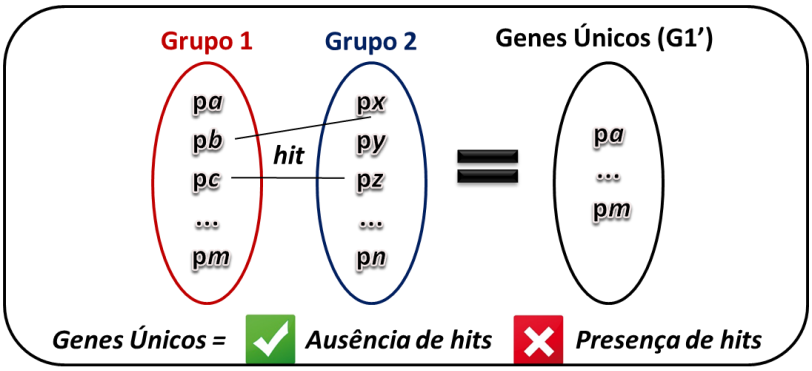

Figura 32. Teoria dos conjuntos: identificação de genes únicos

Um passo a passo é esboçado abaixo:

1. Definir o *taxonomy_Id* do organismo selecionado;
2. Procurar as sequências genômicas na entidade *GenomicSequence* que pertencem ao táxon selecionado. Como resultado, teremos um conjunto de sequências genômicas;
3. Para cada elemento do conjunto de sequências genômicas, devemos encontrar o conjunto de proteínas relacionadas;
4. Por fim, dentre este conjunto de proteínas, devemos selecionar aquelas que não produzem *hit* com proteínas de outro grupo.

Mais uma vez podemos usufruir das abstrações presentes no nosso modelo. Como temos modelado o *taxonomy_id* nas proteínas, podemos buscar diretamente as proteínas sem precisar consultar antes as sequências genômicas, eliminando assim o passo 2.

Analisando a sequência de passos apresentada acima e o modelo lógico apresentado na Seção 4.2.2, primeiramente precisamos encontrar o conjunto de proteínas relacionadas ao organismo em questão (grupo 1). Para isto utilizaremos uma função similar à "*getCountProteinTaxonomy*". A única diferença é que ao invés de retornar a quantidade de proteínas relacionadas a um *taxonomy_Id*, esta deve retornar uma lista das proteínas (*fiocruzid*). A Figura 33 representa esta função.

O passo seguinte é encontrar o conjunto de proteínas relacionadas ao outro organismo (grupo 2), utilizando a função *getProteinTaxonomy* passando como parâmetro o *taxonomy_Id* deste outro organismo.

- Name: *getProteinTaxonomy*

- Input: *integer - taxonomy_Id*

- Output: *numeric collection - fiocruzid*

- Description: *get the fiocruzid collection belonging to a specific taxonomy_Id.*

```
CREATE OR REPLACE FUNCTION getProteinTaxonomy (INTEGER)
RETURNS SETOF NUMERIC AS
$$
SELECT (p.fiocruzid) FROM protein p
WHERE p.taxonomy_id IN (select * from
gettaxonomyidchildrenset($1));
$$
LANGUAGE "sql" IMMUTABLE;
```

Figura 33. Função getProteinTaxonomy.

Após, precisamos identificar se alguma proteína do organismo de pesquisa (grupo 1) possui similaridade com as proteínas dos demais organismos (grupo 2), gerando assim um terceiro grupo (grupo 3). Para isso devemos analisar os atributos *query_fiocruzid e subject_fiocruzid* da tabela de similaridade (hit_pp) para encontrar ocorrências de registros envolvendo a relação entre estes dois grupos. A Figura 34 apresenta a função *getSimilarProtein* responsável por este procedimento.

Por fim, devemos verificar quais proteínas não realizaram alinhamento com qualquer outra proteína do grupo vizinho. Para isto, basta eliminarmos do conjunto de proteínas do organismo origem (grupo 1) as proteínas que possuem similaridade com o outro organismo (grupo 3). A função que representa a identificação de genes únicos é apresentada abaixo (Figura 35).

- Name: *getSimilarProtein*

- Input: *integer - taxonomy_Id*

integer - taxonomy_Id

- Output: *numeric collection - fiocruzid*

- Description: *get the similar fiocruzid collection belonging to a specific taxonomy_Id compared with other organism.*

```
CREATE OR REPLACE FUNCTION getSimilarProtein (INTEGER,
INTEGER) RETURNS SETOF NUMERIC AS
$$
DECLARE
   org1 ALIAS FOR $1;
   org2 ALIAS FOR $2;
BEGIN
   RETURN QUERY
      (SELECT query_fiocruzid FROM hit_pp
      WHERE query_fiocruzid IN (
         select * from explode_array(ARRAY(select * from
         getproteintaxonomy(org1)))) AND
         subject_fiocruzid IN (select * from
         explode_array(ARRAY(select * from
         getproteintaxonomy(org2)))))
      UNION
      (SELECT subject_fiocruzid FROM hit_pp
      WHERE query_fiocruzid IN (
         select * from explode_array(ARRAY(select * from
         getproteintaxonomy(org2)))) AND
         subject_fiocruzid IN (select * from
         explode_array(ARRAY(select * from
         getproteintaxonomy(org1)))));
END
$$
LANGUAGE plpgsql IMMUTABLE RETURNS NULL ON NULL INPUT;
```

Figura 34. Função getSimilarProtein

Esta função, e aquelas que a compõem, pode ser estendida utilizando como parâmetro de entrada a informação do *e-value* desejado, ou algum outro atributo de hit_pp, para restringir a busca e obter resultados mais precisos.

- Name: *getSingleGene*

- Input: *integer - taxonomy_Id*

- Output: *numeric collection - fiocruzid*

- Description: *gets single genes in a taxonomic group.*

```
CREATE OR REPLACE FUNCTION getSingleGene(INTEGER, INTEGER)
RETURNS SETOF NUMERIC AS
$$
DECLARE
   org1 ALIAS FOR $1;
   org2 ALIAS FOR $2;
BEGIN
   RETURN QUERY
      (SELECT getproteintaxonomy (org1))
          EXCEPT
      (SELECT getsimilarprotein(org1,org2));
END
$$
LANGUAGE plpgsql IMMUTABLE RETURNS NULL ON NULL INPUT;
```

Figura 35. Função getSingleGene

4.2.3.2.2. Genes Homólogos

Homologia é o estudo biológico das semelhanças entre estruturas de diferentes organismos que possuem a mesma origem *ontogenética* e *filogenética*. Tais estruturas podem ou não ter a mesma função. Todos os genes, de uma forma ou outra, são homólogos, uma vez que podemos traçar a história evolutiva de todos os organismos até o ancestral comum dos seres vivos. A homologia de genes pode ser de duas formas:

- **Ortólogos** - genes encontrados em diferentes táxons que, ao serem comparados, são passíveis de rastreamento até os eventos que levaram à especiação;
- **Parálogos** - genes em iguais ou diferentes táxons relacionados a ocorrências de duplicação gênica.

Ortologia e paralogia são conceitos chave da genômica evolutiva [Koonin 2005]. A Figura 36 apresenta um exemplo de homologia apresentado por Russo [Russo 2009]. Dado um gene “x”, que, a partir de um evento de duplicação gênica, passa a apresentar duas cópias, “x” e “y”. Tais cópias, x e y, são chamadas de cópias parálogas do mesmo gene x. Supondo que, ao longo

tempo, essa população passe por um evento de especiação, as duas cópias x e y irão evoluir independentemente nas duas espécies, acumulando substituições únicas e, por conseguinte, diferenciando-se. Logo, as cópias resultantes x', x" e y', y" serão cópias ortólogas.

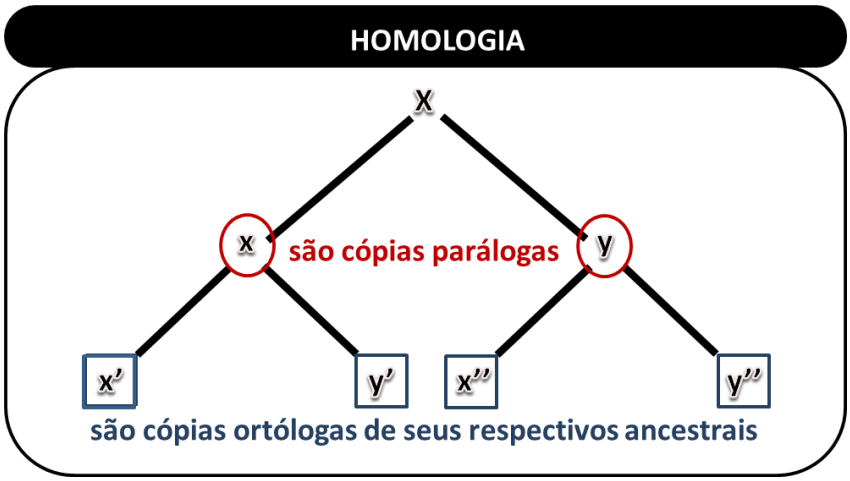

Figura 36. Homologia: representação de cópias parálogas e ortólogas

Na ortologia o objetivo é identificar proteínas a partir de um proteoma completo de um dado organismo que são similares a outras proteínas de diferentes proteomas. A identificação de genes ortólogos segue a mesma lógica de genes únicos. A diferença é que em genes únicos o objetivo é encontrar aquelas proteínas que não possuem similaridade, ao contrário da ortologia que busca exatamente as proteínas semelhantes.

Sendo assim, a função que retorna genes ortólogos é semelhante à função *getSimilarProtein*. A diferença está relacionada basicamente ao tipo de informação de retorno. Abaixo é representada a função que retorna genes ortólogos de uma determinada espécie com relação à outra espécie, bastando informar apenas o *taxonomy_id* de cada espécie. Lembramos que esta função pode ser estendida acrescentando parâmetros de entrada para restringir o universo de dados selecionados, como por exemplo, parâmetros de *hit_pp* e reduzir a busca a um subconjunto de genes. Da mesma forma, o dado de saída pode ser enriquecido com informações a respeito dos genes ortólogos. Uma representação desta função pode ser observada na Figura 37.

Já na paralogia o objetivo é identificar proteínas a partir de um proteoma completo de um dado organismo que são similares a outras proteínas deste mesmo proteoma. Neste caso, a busca deve começar a partir da "raiz" do nodo taxonômico, recebendo o *taxonomy_id* da mesma forma que foi feito para genes únicos. Então, novamente, para este *taxonomy_id* devemos considerar todas as

sequências genômicas a partir da entidade *Genomic Sequence* que são origem de proteínas que tenham sido comparadas no projeto para obter as proteínas.

```
- Name: getOrthologousGene
- Input: integer - taxonomy_Id
         integer - taxonomy_Id
- Output: numeric collection - fiocruzid
- Description: get the orthologous genes to a specific taxonomy_Id.
CREATE OR REPLACE FUNCTION getOrthologousGene (INTEGER,
INTEGER) RETURNS SETOF NUMERIC AS
$$
DECLARE
   org1 ALIAS FOR $1;
   org2 ALIAS FOR $2;
BEGIN
   RETURN QUERY
      (SELECT subject_fiocruzid FROM hit_pp
      WHERE query_fiocruzid IN (
         select * from explode_array(ARRAY(select * from
         getproteintaxonomy(org1)))) AND
         subject_fiocruzid IN (select * from
         explode_array(ARRAY(select * from
         getproteintaxonomy(org2)))))
      UNION
      (SELECT subject_fiocruzid , query_fiocruzid FROM
hit_pp
      WHERE query_fiocruzid IN (
         select * from explode_array(ARRAY(select * from
         getproteintaxonomy(org2)))) AND
         subject_fiocruzid IN (select * from
         explode_array(ARRAY(select * from
         getproteintaxonomy(org1)))));
END
$$
LANGUAGE plpgsql IMMUTABLE RETURNS NULL ON NULL INPUT;
```

Figura 37. Função getOrthologousGene

Por conta da abstração presente no modelo, que apresenta o *taxonomy_id* nas proteínas, podemos obter o conjunto de proteínas sem precisar percorrer as sequências genômicas.

Por fim, devemos procurar na tabela hit_pp a existência de similaridade entre as proteínas deste proteoma. A Figura 38 apresenta a função para obtenção de genes parálogos.

```
- Name: getParalogousGene
- Input: integer - taxonomy_Id
- Output: numeric collection - fiocruzid
- Description: get the paralogous genes to a specific taxonomy_Id.
CREATE OR REPLACE FUNCTION getParalogousGene(INTEGER)
RETURNS SETOF NUMERIC AS
$$
DECLARE
   org ALIAS FOR $1;
BEGIN
   RETURN QUERY
     (SELECT subject_fiocruzid FROM hit_pp
     WHERE query_fiocruzid IN (select * from
        explode_array(ARRAY(select * from
        getproteintaxonomy(org)))) AND
        subject_fiocruzid IN (select * from
        explode_array(ARRAY(select * from
        getproteintaxonomy(org)))));
END
$$
LANGUAGE plpgsql IMMUTABLE RETURNS NULL ON NULL INPUT;
```

Figura 38. Função getParalogousGene.

Da mesma forma genes ortólogos, a função de genes parálogos pode ser estendida acrescentando parâmetros de entrada para restringir o universo de dados selecionados, como por exemplo, parâmetros de hit_pp.

4.2.4. Execução

Para questões de desempenho, o computador utilizado para este estudo de caso segue a seguinte configuração de software e hardware:

- Software:
  - LINUX Kernel version 2.6.26 – Debian;
  - PostgreSQL 9.1.
- Hardware:
  - INTEL core 2 Quad Q6600;
  - 2.4 GHz;
  - 4 GB de RAM;
  - 4 TB de HD não redundante - 3 x 1,36 TB.

4.2.4.1. Carga dos Dados

Para a realização dos testes referentes às consultas/funções implementadas, primeiramente foi realizada a carga dos dados originados de diferentes fontes:

- Dados de comparação (hits) – Fiocruz, através do projeto de comparação de genomas em parceria com o *World Community Grid*;
- Taxonomia – NCBI *Taxonomy*;
- Dogma central – NCBI *Reference Sequence* (RefSeq)
- Anotações – as anotações são oriundas das seguintes fontes:
    - Domínio – Pfam;
    - Enzimas – KEGG;
    - Ontologias – Gene Ontology.

Para facilitar o processo de carga, foram desenvolvidos *scripts* na linguagem *Perl* para a carga dos dados no banco de dados relacional. No entanto, durante este procedimento foram enfrentados alguns problemas relacionados à quantidade de dados das fontes de origem, ou seja, a quantidade de dados que deveria ser processada e transformada em sentenças INSERT para o banco de dados era muito grande. Uma alternativa foi utilizar o comando COPY, que é exatamente um comando para carregar grandes quantidades de dados a partir de arquivos texto puros. Embora não faça diferença para nosso caso, a única desvantagem em relação ao comando INSERT é possuir menos flexibilidade.

Após a carga dos dados de proteínas, sequência genômica, cds e taxonomia, foi realizada a carga dos hits, que somados em relação à espaço físico para armazenamento ocupam 360GB (compactado). Esses dados estão organizados em 910 arquivos no formato TAR (empacotado). Cada arquivo, na verdade é uma lista de 2000 arquivos no formato *.TAR.GZ.

Para este estudo de caso, foi utilizado apenas um subconjunto dos dados de similaridade. Este subconjunto contempla apenas os dados de similaridade cujo valor de *e-value* é menor ou igual a 1.0e-3 (valor de corte). Utilizamos este valor de corte, pois este é o valor base de similaridade assumido entre as proteínas do PWD [Otto et. al. 2010]. Além disso, quando carregamos o conjunto completo, tivemos problemas de espaço em disco para o processo de criação de índices. Com a utilização deste filtro, a quantidade de dados existentes na tabela hit_pp passou de 900 GB para aproximadamente 288 GB.

Inicialmente a carga destes dados foi realizada através de sentenças INSERT. Porém, da mesma forma que anteriormente, enfrentamos dificuldades para inseri-los no banco, dado o enorme volume de dados apresentado. A solução foi utilizar sentenças COPY ao invés de INSERT. Mesmo aplicando estas regras e condições, a carga dos dados de similaridade demorou mais de 24 horas, considerando que o processamento era exclusivo para este processo.

Durante o processo de carga dos hits, enfrentamos inúmeros problemas, que vão desde a falta de energia até problemas de inconsistência/ruídos em alguns dados, que ocasionaram a interrupção/falha do processo. Deste modo, para a importação dos hits, foi adotada uma política de carga em ondas/partes. O objetivo foi dividir e diminuir o conjunto de dados em cada arquivo para facilitar a busca e correção de erros, e o retrabalho do processo de carga, no caso de ocorrerem falhas. Para tanto, o arquivo contendo os comandos COPY foi divido em 10 arquivos temporários, contendo dados de 200 arquivos *.tar.gz cada.

Para o estudo de caso desta Tese, não foram carregados os dados referentes à ORFs e às anotações de ontologia. Isto porque, reduzimos o domínio de pesquisa apenas para proteínas e o conjunto de anotações para domínios e enzimas. Abaixo, a Tabela 3 apresenta o espaço físico ocupado por cada tabela do banco PWD.

Tabela 3. Relatório do espaço ocupado pelas tabelas do PWD

TABELA	TAMANHO
Cds	617 MB
Domain	936 kB
Domain_annotation	1238 MB
Enzyme	808 kB
Enzyme_annotation	9064 kB
Gene	512 MB
Gene_ontology	0 bytes
Genomic_sequence	413 MB
Goannotation	0 bytes
Hit_oo	0 bytes
Hit_op	0 bytes
Hit_pp	288 GB
Orf_region	0 bytes
Orf_t	0 bytes
Protein	1248 MB
Synonimum	22 MB
Tax_rank	8192 bytes
Taxonomy	59 MB

Analisando os dados da Tabela 3 podemos observar que, em termos de espaço físico ocupado, as maiores tabelas, que compõem o dogma central da biologia molecular, são representadas por: hit_pp, protein, cds, gene e genomic_sequence, nesta ordem.

4.2.4.2. Implementação e Teste das Consultas

Após a carga dos dados foi dado início à implementação de algumas consultas, conforme apresentado na Seção 4.2.3. Para este conjunto inicial de testes foi utilizado o banco de dados relacional com a utilização de índices para otimização e processamento de consultas, e chaves para integridade referencial.

A utilização de índices e chaves (primária e estrangeira) são de grande importância para auxilar do ganho de desempenho de consultas e integridade referencial dos dados, respectivamente. Por esta e outras características que muitos defendem o uso de SGBDs para armazenar e gerenciar dados biológicos, uma vez que já existem mecanismos consolidados para o armazenamento e gerência de dados. No entanto, a discussão acaba se voltando para a expressividade dos tipos de dados existentes, quando utilizado para o domínio biológico.

Vale destacar o processo realizado para a criação de índices sobre a tabela *hit_pp*. Como as consultas que envolvem dados de similaridade referenciam sempre os atributos *query_fiocruzid* e *subject_fiocruzid*, é natural que sejam criados índices para estes dois atributos, além do que são referências à tabela de proteínas.

Com a carga inicial total dos dados de hits de proteínas (900 GB), tínhamos aproximadamente 250 GB de espaço em disco livre. No entanto esta quantidade não foi suficiente para a criação de *query_fiocruzid*. Levando em consideração que também tínhamos que criar o índice para *subject_fiocruzid*, decidimos filtrar os dados de similaridade para *e-value* menor ou igual a 1.0e-3 (valor de corte), conforme adotado por [Otto et. al. 2010]. Mesmo reduzindo o conjunto de dados de similaridade, a criação de cada índice levou aproximadamente 24 horas.

Outra técnica utilizada foi a clusterização da tabela hit_pp com base nesses índices. Resumindo as ações adotadas para construção deste cenário para os testes, temos:

Cenário:

- Banco de dados contendo um *subset* da tabela hit_pp;
- Criação de índices nas colunas *query_fiocruzid* e *subject_fiocruzid*, para acelerar o desempenho das consultas, ambos da tabela hit_pp;
- Clusterização da tabela hit_pp com base nos seus índices;
- Criação das chaves primárias (*primary key* - PK) e restrições de unicidade nas demais tabelas.

Para a realização dos testes utilizamos dois organismos como base para a realização das consultas, ambos do gênero *Xanthomonas* (*Taxonomy id: 338*), que são:

- Xanthomonas axonopodis pv. citri str. 306 chromosome, complete genome.
    - gbkid = NC_003919.1;
    - Taxonomy id = 190486;
    - GI = 21240774;
    - Refseq Protein = 4427.
- Xanthomonas campestris pv. campestris str. ATCC 33913, complete genome.
    - gbkid = NC_003902.1;
    - Taxonomy id = 190485;
    - GI = 21229478;
    - Refseq Protein = 4179.

Primeiramente foram criadas e testadas as consultas ditas “básicas” (Seção 4.2.3.1), pois estão relacionadas apenas a questões estatísticas dos dados. Após foram criadas e testadas as consultas “complexas” (Seção 4.2.3.2). A Tabela 4 apresenta um resumo dos resultados obtidos.

Tabela 4. Resultado dos testes.

CONSULTA	RESULTADO	TEMPO (ms)
Básicas		
Quantidade de Proteínas Comparadas	1947724	985.290
Proteínas com Sequência Genômica de Origem	8744479	2,359.135
Quantidade de Genomas que Pertencem a um Grupo Taxonômico (taxonomy id: 338)	22	19.888
Quantidade de Proteínas que Pertencem a um Grupo Taxonômico (taxonomy id: 338)	42923	39.116
Quantidade de Hits para uma Proteína “X”, dado uma Linha de Corte (fiocruzid: 21264186; evalue: 1.0e-5)	3	3.637
Complexas		
Genes Únicos (taxonomy id: 190486; 190485)	585 linhas	5,761.098
Genes Ortólogos (taxonomy id: 190486; 190485)	3806 linhas	5,862.703
Genes Parálogos (taxonomy id: 190486)	12655 linhas	3,450.374

Analisando os resultados obtidos das consultas, podemos verificar que todos eles tiveram um desempenho excelente. Mesmo as consultas que fazem referências à tabela de hit_pp, que contém o maior número de registros, teve resultado satisfatório. Esse resultado é fruto da utilização de índices na busca pelos resultados.

Podemos exemplificar isso analisando a função *getSimilarProtein (INTEGER, INTEGER)*. Inicialmente, a comparação entre query_fiocruzid e subject_fiocruzid com as proteínas do organismo destino era realizada diretamente sobre o resultado da função *getproteintaxonomy(INTEGER)*. No entanto, analisando o plano de consulta do otimizador do SGBD, foi constado que nesta parte da consulta não era utilizado índice e sim um *full scan* na tabela hit_pp. Em termos de desempenho, a consulta levava horas para gerar o resultado. A solução foi converter o resultado da função *getproteintaxonomy(INTEGER)* para um *array*. Desta forma o otimizador gerou um plano de consulta utilizando índice neste ponto da consulta e como resultado o tempo de resposta caiu para a casa dos milésimos de segundo.

4.3. Considerações Finais

A implementação do modelo de dados e a execução das funções utilizando um SGBD relacional demonstrou como o uso do mesmo pode auxiliar na organização dos dados e facilitar e otimizar o processo de pesquisa e obtenção de resultados. Ficou também evidenciado que o uso de SGBDs só traz ganhos quando utilizado mecanismos, inerentes da própria tecnologia, apropriados para indexação e pesquisa.

Outro ponto relevante é com relação ao volume de dados. Sabemos que a quantidade de dados biológicos cresce exponencialmente. Desta forma, devemos pensar em mecanismos apropriados para a carga e atualização desta massa de informação, de modo que acompanhe este crescimento. Com relação ao armazenamento, gerência e processamento destes dados, já existem abordagens, tais como, particionamento de tabelas, SGBDs distribuídos e clusters de computadores. Todas estas abordagens são abstratas ao usuário final, devendo apenas decidir qual é a abordagem mais adequada a cada situação.

Além disso, a implementação do esquema lógico-relacional e a realização das consultas sobre o mesmo evidenciou a eficácia do modelo conceitual proposto. Da mesma forma, a implementação do tipo abstrato de dados "*bio-string*" ilustrou que a dificuldade em gerenciar dados sequenciais biológicos não é um problema do modelo relacional, mas sim da falta de semantica nas estruturas de dados existentes.