MySQL – BANCO DE DADOS PARA ORGANIZAR E INTERPRETAR SEUS DADOS

O volume de dados gerados em projetos de Bioinformática é geralmente muito grande. Por isso se faz necessário o uso de ferramentas não somente para armazená-los, mas também para inter-relacioná-los e interpretá-los de uma forma automatizada e em larga escala. Essas ferramentas são conhecidas como bancos de dados e possuem uma série de funções que permitem acessar, correlacionar e visualizar os dados da maneira mais relevante para o pesquisador.

Um sistema que gerencia bancos de dados é chamado de SGBD (Sistema Gerenciador de Banco de Dados). Há vários SGBDs disponíveis, cada qual direcionado a um tipo de expectativa do usuário e do volume de dados a tratar. O SGBD MySQL [1] é simples, porém muito eficiente, além de ser distribuído gratuitamente. Por essa razão é um dos mais utilizados em projetos de Bioinformática.

Bancos de dados são estruturas de dados compostas basicamente por tabelas. Existem tabelas internas do sistema e tabelas chamadas de negócio. Usuários criam as tabelas de negócio para inter-relacionar entidades. Tabelas são nada mais que planilhas eletrônicas: possuem linhas e colunas. A sintaxe do MySQL permite escolher colunas para pesquisar e comparar entradas (registros) das linhas da(s) tabela(s) de negócio. Enquanto isso, as tabelas de sistema cuidam da indexação e dos relacionamentos entre campos de tabelas distintas.

Inicialmente veremos como poderíamos proceder para analisar resultados de BLAST [2] sem o uso de MySQL. Vamos analisar um *megablast* que utiliza 25 mil CDS humanos como *query* e 500 mil transcritos de tumor de mama como *database*.

Este tutorial é feito para ser executado em Linux ou MacOS. Se você só possui acesso à plataforma Windows, use o *cygwin* [3].

Preparação

1. Crie uma pasta local denominada 'mysql_aula'. Os arquivos deste tutorial serão copiados para essa pasta e a partir dela os comandos serão executados.
2. Faça o download dos dois arquivos abaixo e copie-os para a pasta *mysql_aula* criada acima:
   - http://biodados.icb.ufmg.br/tutorial2011/MySQL/tumor.zip
   - http://biodados.icb.ufmg.br/tutorial2011/MySQL/cds.zip
3. Descompacte os arquivos baixados utilizando o programa unzip:
   - unzip tumor.zip
   - unzip cds.zip
4. Agora você também terá no seu diretório os arquivos tumor.seq e h.sapiens.nuc, que correspondem aos FASTAs de tumor de mama e CDS humanos, respectivamente. Se desejar, você pode apagar os arquivos .zip nesse ponto, no intuito de economizar espaço em disco.
   - rm *.zip
5. O arquivo tumor.seq precisa agora ser formatado utilizando a ferramenta do *toolkit* BLAST denominada *formatdb* [4]. Esse passo é necessário, pois os programas BLAST só aceitam *databases* formatadas. Para formatar tumor.seq, rode o comando:
   - formatdb -i tumor.seq -p F -o T

     Legenda:

     -i = input

     -p F = define que o input NÃO é de proteínas, e sim, de nucleotídeos

     -o T = cria índices
6. Após a execução de formatdb você perceberá que uma série de arquivos iniciados com prefixo 'tumor.seq' foram criados. Não se preocupe com eles, pois os mesmos são utilizados apenas pelo BLAST durante as análises. Apenas deixe-os onde estão.
7. Nesse ponto você possui os arquivos necessários para a execução dos programas BLAST.

Execute megablast sobre o conjunto de dados

Agora vamos utilizar a ferramenta BLAST conhecida como MEGABLAST. Essa ferramenta permite a busca por alto grau de similaridade entre sequências. Recebe como entrada um FASTA que denominamos *query*, e também a *database* onde as sequências *query* serão alinhadas. Em nosso caso, utilizamos o FASTA de CDS humanos (h.sapiens.nuc) como *query*, e a base contendo sequências de tumor de mama como *database.* Os alinhamentos revelam transcritos de cada um dos 25 mil genes. Alguns não são transcritos e não aparecerão no resultado. Outros aparecerão genes (CDS) aparecerão na coluna 1 do resultado muitas vezes, são os mais expressos.

Execute o comando abaixo a partir do diretório *mysql_aula,* criado anteriormente.

- megablast -i h.sapiens.nuc –d tumor.seq –D 3 –F F –a 4 –p 96 –s 100 –o megakegg

  Legenda:

  -i = input

  -d = database

  -D 3 = saida tabulada

  -F F = filtro de baixa complexidade desligado (F = False)

  -a 4 = use 4 processadores

  -p 96 = mínima identidade 96%
  (seqüenciamento pode ter até 4% de erro)

  -s 100 = mínimo escore 100

  -o = nome do arquivo de saída

O comando gera o arquivo 'megakegg' como saída. Esse arquivo está em formato tabulado e suas linhas contêm os CDS que tiveram bom alinhamento com sequências da base de tumor de mama.

Interprete o resultado com comandos de Shell

Vamos agora utilizar alguns comandos de Shell para analisar o resultado gerado pelo MEGABLAST. Execute o comando abaixo:

- cat megakegg | awk ‘{print $1}’ | sort | uniq –c | sort –k 1,1 –n –r > mais_hits

  Legenda:

  cat = traz para a memória o conteúdo de megakegg

  | = (pipe) acopla uma nova tarefa à anterior

awk = nenhuma condição é exigida (nada após as aspas) e ação de imprimir a coluna 1 (entre as chaves)

sort = ordena as *queries*

uniq = mostra cada *query* uma única vez

uniq –c = idem, mas mostra quantas de cada uma haviam

sort –k 1,1 = ordena pela coluna 1, que é o número de ocorrências das *queries* (objetivo de contar quantos transcritos cada CDS tem)

sort –n = entende valores como números

sort –r = reverso, ordena do maior para o menor

O comando permite visualizar o número de *hits* de cada CDS no arquivo de saída do MEGABLAST. Além disso, o resultado vem ordenado, mostrando primeiro aqueles CDS mais expressos, que tiveram mais *hits*. Para mais informações sobre os programas utilizados acima, consulte seus manuais (ex: man awk, man uniq, man sort, etc).

Selecione algum hit e procure a identificação no arquivo h.sapiens.nuc

- cat h.sapiens.nuc | grep hsa:6728

Através de *grep*, é possível encontrar uma linha em um arquivo que possui o texto especificado. Assim, o comando acima encontra a descrição do CDS *hsa:6728* no arquivo FASTA que serviu como *query* para MEGABLAST *(*arquivo h.sapiens.nuc).

O resultado é:

>hsa:6728 SRP19; signal recognition particle 19kDa ; K03105 signal recognition particle subunit SRP19

Observando o arquivo 'mais_hits' gerado na etapa anterior, observamos que o gene hsa:6728 é o CDS que mais gerou hits com o *database* de tumor de mama (total de 1386 *hits*).

Através da análise executada acima, pudemos caracterizar a expressão de um gene em tumores de mama. Mas imagine que quiséssemos identificar todos os genes. Seria muito custoso efetuar esse procedimento com cada gene. Por isso iremos apresentar uma introdução ao MySQL. O passo inicial é a preparação dos arquivos e tabelas.

Utilizando MySql para a análise de resultados do BLAST

Com o uso de bancos de dados, a análise do resultado do MEGABLAST fica muito mais potente. Os bancos de dados (inclusive o MySql) geralmente utilizam a linguagem

universal SQL (*Structured Query Language)* para consultas e operações. SQL possui sintaxe simples, muito próxima do inglês, o que permite buscas complexas em uma ou mais tabelas. Além disso, bancos de dados possuem sistemas de indexação que permitem a obtenção de resultados de uma maneira muito mais rápida do que a leitura de arquivos texto linha a linha. No passo a passo a seguir, veremos como fazer a análise de resultados do BLAST utilizando o banco de dados MySQL. Antes, porém, iremos preparar o banco de dados para uso.

Instalação e preparação do MySQL

O MySQL está presente na maioria das distribuições Linux. É necessário que o mesmo esteja instalado em sua máquina para prosseguir com o tutorial. Caso não esteja instalado, favor verificar como proceder em http://www.mysql.com.

Você precisará também da senha de *root* do MySql para criar um usuário e um banco de dados para você. Se não tiver permissão, peça ao administrador de seu sistema para fazer isso para você. (por padrão, após a instalação, o usuário *root* do MySQL não possui senha. Portanto, tente senha vazia).

Logue como root no MySQL utilizando o comando abaixo:

- mysql -u root -p

ou

- mysql -u root (se o usuário root não possui senha)

Após logar, você verá o *prompt* de comando do MySQL, como a seguir:

mysql>

É nesse *prompt* que todas as consultas e operações SQL no banco são digitadas.

Vamos agora criar um banco de dados para armazenar os resultados de nosso BLAST. Digite (no *prompt* do MySQL):

mysql> create database tumor;

Com esse comando, você criou uma *database* com o nome "tumor". Essa *database* por enquanto está vazia, ou seja, não possui tabelas. Serão criadas, posteriormente. Note que colocar um ponto-e-vírgula ao final dos comandos MySQL é essencial.

Agora vamos criar um usuário com permissões de acesso ao banco criado. Digite o seguinte comando:

mysql> grant all privileges on tumor.* to tutorial@localhost identified by 'tutorial123';

Com esse comando, você fez duas coisas:

1) Criou um usuário *MySQL* com o nome "tutorial" e senha "tutorial123".

2) Autorizou o usuário recém-criado a consultar e modificar o banco de dados denominado "tumor" e também todas as usas tabelas.

Você pode agora sair do ambiente MySQL utilizando o comando:

mysql> exit

Você está agora preparado para iniciar o passo a passo de análise de resultados do BLAST usando MySQL, que se inicia na próxima seção.

Análise de resultados do BLAST usando MySQL

1. Para esta parte do tutorial você precisará de alguns arquivos. Faça o *download* do arquivo *mysql_tutor.zip* em:
   http://biodados.icb.ufmg.br/tutorial2011/MySQL/mysql_tutor.zip

Copie o arquivo baixado para sua pasta *mysql_aula e descompacte-o.* Note que o arquivo *megakegg* está nesse pacote. Esse arquivo contém o resultado do *megablast* executado na seção anterior, por isso você já deve tê-lo em sua pasta. Não é necessário sobrescrevê-lo ao descompactar o arquivo *zip*, já que se trata exatamente do mesmo arquivo. Os demais arquivos descompactados serão usados posteriormente.

2. Crie um arquivo tabulado com apenas algumas colunas do *blast*:
   - cat megakegg | awk -v OFS="\t" '($1 != "#") {print $1,$2,$3,$11,$12}' > megakegg_tab

     Legenda:

     -v = OFS insere um tabulador (\t) no arquivo de saída, entre as colunas selecionadas

O comando acima selecionou as colunas 1, 2, 3, 11 e 12 do arquivo *megakegg* e também eliminou as linhas cuja primeira coluna começa com "#" (linhas começadas com esse caractere são comentários). O novo arquivo foi gravado com o nome *megakegg_tab.*

3. Acesse o MySQL utilizando o nome de usuário e senha criados na seção anterior (tutorial e tutorial123, respectivamente).
   - mysql -u tutorial -p

   Legenda:

   -u = nome do usuário a logar

   -p = informa que você vai digitar uma senha de usuário

Note que o sistema pedirá a senha após você teclar ENTER. Digite a senha (observe que o cursor não se move enquanto você digita) e tecle ENTER novamente.

4. Neste ponto você verá novamente o *prompt* MySQL, e estará logado como o usuário *tutorial*. Vamos listar as *databases* que o usuário pode acessar. Para isso, entre com o comando:

mysql> show databases;

Lembre-se: todos os comandos MySQL devem terminar com ponto e virgula (;). Você verá uma lista com o nome dos bancos de dados disponíveis, como abaixo.

```
+--------------------+
| Database           |
+--------------------+
| information_schema |
| tumor              |
+--------------------+
```

Observe que o usuário *tutorial* possui acesso a duas *databases*. A primeira é uma *database* de sistema, com a qual você não deve se preocupar. O segundo é a *database* denominada *tumor*, que foi criada na seção anterior.

5. A *database tumor* será utilizada nesse tutorial. Você deve informar ao sistema que deseja utilizar essa *database*. Para isso, entre com o comando abaixo:

mysql> use tumor;

O sistema então informa que utilizará essa *database* para efetuar as operações e consultas inseridas.

6. Agora vamos verificar as tabelas presentes na *database tumor*. Utilize o seguinte comando:

mysql> show tables;

O resultado da execução é

Empty set (0.00 sec)

Isso mostra que nossa *database* ainda está vazia. No próximo passo, vamos criar nossa primeira tabela.

7. Utilize o próximo comando para criar uma tabela para armazenar os dados selecionados do BLAST (arquivo *megakegg_tab*):

mysql> create table result_blast (cds varchar(15), subject varchar(50), identity double(5,2), evalue varchar(10), score int, index cds_idx (cds));

Com o comando acima, criamos uma tabela chamada *result_blast* que possui 5 colunas (*cds*, *subject*, *identity*, *evalue* e *score)*. Os valores que vem logo após o nome das colunas determinam o tipo de dados que a coluna armazenará. Uma breve descrição de cada tipo de dados vem abaixo:

Legenda:

- varchar(N) = coluna tipo texto, com no máximo N caracteres.
- double(M,N) = coluna do tipo decimal, com M digitos no total e N casas decimais. No caso, double(5, 2) comporta 100.00
- int = coluna do tipo número inteiro

Como vemos, os tipos de cada coluna são definidos de acordo com o tipo de dados que se receberá do arquivo resultado do *BLAST*, a ser lido posteriormente. O comando *index cds_idx (cds)* adiciona um índice na coluna *cds*, para facilitar as consultas.

8. Sempre que quiser, você pode verificar as tabelas criadas com o comando:

mysql> desc result_blast;

Você verá uma tela como a próxima. Ela descreve a tabela, identificando os nomes e tipos das colunas.

```
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| cds      | varchar(15) | YES  | MUL | NULL    |       |
| subject  | varchar(50) | YES  |     | NULL    |       |
| identity | double(5,2) | YES  |     | NULL    |       |
| evalue   | varchar(10) | YES  |     | NULL    |       |
| score    | int(11)     | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
```

9. Agora vamos carregar o resultado do *megablast* para a tabela recém-criada. Para isso, utilize o comando abaixo:

mysql> load data local infile 'megakegg_tab' into table result_blast;

Nota: se vc não entrou no MySQL a partir do diretório onde megakegg_tab está, deve colocar o caminho completo dele, como: '/home/vc/negakegg_tab'.

O comando *load data local infile* carrega um arquivo texto tabulado para uma tabela do banco.

10. Verifique o conteúdo da tabela *result_blast* depois do carregamento do arquivo:

mysql> select * from result_blast limit 10;

O comando *select* é provavelmente o mais utilizado em operações no MySQL. Ele permite uma consulta a uma ou mais tabelas, com a utilização opcional de filtros específicos. Nesse caso, selecionamos todas as colunas (*) da tabela *result_blast* e limitamos o resultado aos 10 primeiros registros (cláusula *limit 10).* O seu resultado será como abaixo:

```
+------------+----------------------+----------+--------+-------+
| cds        | subject              | identity | evalue | score |
+------------+----------------------+----------+--------+-------+
| hsa:5701   | lcl|000079_1820_0215 |   100.00 | 7e-68  |   260 |
| hsa:55255  | lcl|000457_0385_0605 |    97.90 | 1e-118 |   428 |
| hsa:23412  | lcl|000520_0240_2623 |    98.54 | 5e-67  |   256 |
| hsa:55744  | lcl|000560_1868_2572 |    98.56 | 4e-107 |   389 |
| hsa:79020  | lcl|000907_0864_1433 |   100.00 | 4e-125 |   450 |
| hsa:6122   | lcl|001079_0272_2374 |    99.51 | 1e-109 |   398 |
| hsa:10200  | lcl|001332_0216_1609 |    98.81 | 2e-133 |   476 |
| hsa:10200  | lcl|001412_1052_3310 |   100.00 | 6e-60  |   232 |
| hsa:157313 | lcl|001527_1414_2084 |   100.00 | 2e-70  |   270 |
| hsa:90799  | lcl|001654_1300_0808 |   100.00 | 2e-131 |   472 |
+------------+----------------------+----------+--------+-------+
```

Com este resultado, verificamos que o conteúdo do arquivo *megakegg_tab* agora está em colunas na tabela *result_blast*.

11. É possível verificar o total de registros na tabela utilizando a cláusula *count(*)* logo após o comando *select.* Execute o comando abaixo:

mysql> select count(*) from result_blast;

O resultado será:

```
+----------+
| count(*) |
+----------+
|   254999 |
+----------+
```

O número total de registros na tabela *result_blast* é, portanto, 254999, o qual corresponde ao número de *hits* de nosso *megablast.*

12. Podemos contar também o número de *hits* por CDS, da mesma forma que fizemos anteriormente via linha de comando utilizando *awk* | *sort* | *uniq*. Para isso utilizamos também a cláusula *count(*),* mas dessa vez associada a uma cláusula *group by.* Essa última cláusula agrupa os registros iguais em uma determinada coluna, para fins de contagem, soma de valores e outras operações de grupo. Digite o comando abaixo para descobrir os CDS que deram mais *hits:*

mysql> select *, count(*) from result_blast group by cds order by count(*) desc limit 10;

Verifique que utilizamos a cláusula *order by count(*) desc* ao final do comando. Essa cláusula ordena os resultados, mostrando primeiro aqueles grupos de CDS com o maior número de *hits.* Esses são, portanto, os dez CDS mais expressos na amostra de tumor. Veja abaixo o resultado do comando:

```
+------------+----------------------+----------+--------+-------+----------+
| cds        | subject              | identity | evalue | score | count(*) |
+------------+----------------------+----------+--------+-------+----------+
| hsa:6728   | lcl|181254_3359_2079 |   100.00 | 1e-107 |   390 |     1386 |
| hsa:6206   | lcl|249582_1187_2560 |    96.07 | 5e-103 |   375 |     1314 |
| hsa:11340  | lcl|145491_0244_0456 |    99.59 | 6e-132 |   472 |     1278 |
| hsa:28998  | lcl|239125_3727_0646 |    98.85 | 1e-88  |   327 |     1164 |
| hsa:9521   | lcl|000957_0743_1006 |    99.56 | 3e-123 |   442 |     1154 |
| hsa:539    | lcl|089256_3027_3144 |    97.44 | 2e-94  |   347 |     1124 |
| hsa:10605  | lcl|267166_1358_3876 |    97.44 | 5e-72  |   274 |     1111 |
| hsa:645139 | lcl|309393_2903_3714 |    97.24 | 3e-123 |   444 |     1107 |
| hsa:9232   | lcl|036323_1743_0199 |    99.58 | 1e-129 |   464 |     1059 |
| hsa:6189   | lcl|003783_0982_2141 |   100.00 | 2e-54  |   214 |     1046 |
+------------+----------------------+----------+--------+-------+----------+
```

O resultado mostra que o CDS *hsa:6728,* por exemplo, teve 1386 *hits* no transcriptoma de tumor de mama.

13. Vamos criar agora uma nova tabela contendo o nome do CDS e o número de *hits* do mesmo. Para isso, basta associar um *create* table a um *select*, como abaixo:

mysql> create table hsa_count select cds, count(*) as hits from result_blast group by cds;

A nova tabela criada se chama *hsa_count*, e possui 7091 registros, cada um correspondente a um CDS distinto que obteve *hit* na amostra de tumor. A coluna com a contagem de *hits* foi renomeada para *hits*, utilizando a cláusula *as hits* no comando *select.*

14. Sempre verifique uma tabela criada com:

mysql> select * from hsa_count limit 10;

Agora é hora de ver a descrição dos genes correspondentes a cada símbolo *hsa*, a fim de dar uma interpretação biológica ao resultado. Para isso, vamos criar uma nova tabela e carregar nela o arquivo *hsa_description*, o qual se encontra em sua pasta *mysql_aula*. Note que vamos novamente utilizar os comandos *create table* e *load data local infile.*

15. Execute os comandos na sequencia:

mysql> create table hsa_description (cds varchar(15), description varchar(400), index cds_idx (cds));

mysql> load data local infile '<diretorio mysql_aula>/hsa_description' into table hsa_description;

16. Queremos agora adicionar uma coluna *description* na tabela hsa_count, de modo que fique mais rápido visualizar a descrição do CDS. Mesmo uma tabela já

criada pode ser alterada para conter mais informação. Altere a *hsa_count* para conter também uma coluna de descrição do CDS:

mysql> alter table hsa_count add column description varchar(400);

O comando acima altera a tabela *hsa_count,* adicionando uma nova coluna denominada *description,* de tipo texto, com no máximo 400 caracteres.

17. Para verificar que a nova coluna foi adicionada, porém ainda não contém dados, use:

mysql> desc hsa_count;

select * from hsa_count limit 10;

18. Combinar tabelas é algo comum e muito usado em MySQL. Atualize a tabela *hsa_count* com dados de descrição dos genes, assim é muito mais fácil saber quem é quem:

mysql> update hsa_count, hsa_description set hsa_count.description = hsa_description.description where hsa_count.cds = hsa_description.cds;

O comando *update* serve para atualizar dados em colunas de tabelas e é certamente um dos mais utilizados. Nesse caso, estamos atualizando a tabela *hsa_count* com dados da tabela *hsa_description.* O comando *set* indica que colunas serão atualizadas e o comando *where* é um filtro indicando os registros onde a atualização deve ocorrer. No caso acima, dizemos que queremos copiar o conteúdo da coluna *description* da tabela *hsa_description* para a coluna *description* da tabela *hsa_count*, mas somente nos registros onde as colunas *cds* das duas tabelas são iguais. Em suma, as tabelas são "unidas" pela coluna cds e, nesse ponto, as descrições do CDS que estavam na tabela *hsa_description* são copiadas para a tabela *hsa_count.*

19. Verifique novamente os dados da tabela utilizando *select* e *limit.* Note que agora as descrições de cada gene estão presentes, como mostra o exemplo abaixo.

```
+---------------+------+-----------------------------------------+
| cds           | hits | description                             |
+---------------+------+-----------------------------------------+
| hsa:10001     |    2 | MED6; mediator complex subunit 6        |
| hsa:10006     |   16 | ABI1; abl-interactor 1                  |
| hsa:10011     |   15 | SRA1; steroid receptor RNA activator 1  |
| hsa:100127922 |   23 | similar to mCG21560                     |
| hsa:100127944 |    2 | hypothetical LOC100127944               |
+---------------+------+-----------------------------------------+
```

20. Agora iremos associar nossos CDS a grupos de KOs (grupos de genes ortólogos). Crie uma tabela que relaciona KOs e CDS, com o seguinte comando:

mysql> create table hsa_ko (cds varchar(15), ko varchar(11), hits bigint default 0, index cds_idx (cds), index ko_idx(ko));

Essa tabela, denominada *hsa_ko*, identifica a que KO pertence cada CDS. Criamos também uma coluna *hits* que receberá uma cópia do número de *hits* do CDS na amostra de tumor. Essa coluna é do tipo *bigint*, que significa um inteiro grande, e seu valor padrão é 0.

21. Essa é a terceira tabela que criamos. Para verificar a qualquer momento, as tabelas que existem no banco de dados, basta usar o comando:

mysql> show tables;

No momento, o resultado desse comando será:

```
+-----------------+
| Tables_in_tumor |
+-----------------+
| hsa_count       |
| hsa_description |
| result_blast    |
+-----------------+
```

22. Carregue os dados do arquivo *hsa_ko.list* na tabela recém criada.

mysql> load data local infile '/hsa_ko.list' into table hsa_ko;

Caso o arquivo hsa_ko.list não esteja na pasta de onde o MySQL foi acessado, o caminho complete do arquivo precisa ser dado. O comando acima preencheu as colunas *cds* e *ko* da tabela recém-criada *hsa_ko.* A coluna *hits*, porém, permanece com valores zerados.

23. Preencha essa coluna a partir de dados da tabela *hsa_count*, utilizando o seguinte comando:

mysql> update hsa_ko, hsa_count set hsa_ko.hits = hsa_count.hits where hsa_ko.cds = hsa_count.cds;

24. Verifique o numero de pares CDS x KO:

mysql> select count(*) from hsa_ko;

Note que essa contagem inclui todos os CDS, inclusive aqueles que não tiveram *hits* na amostra de tumor. O total obtido foi de 9475.

25. Vamos excluir pares CDS x KO cujos CDS não obtiveram *hits*, pois não nos interessam:

mysql> delete from hsa_ko where hits = 0;

O comando *delete* apaga de uma determinada tabela os registros que seguem o filtro especificado na cláusula *where.* Nesse caso, exclui todos os registros cujo valor na coluna *hits* é 0.

26. Verifique novamente o numero de pares CDS x KO restantes:

mysql> select count(*) from hsa_ko;

O novo total de registros é 3384, que correspondem aos CDS que tiveram pelo menos um *hit.*

27. Agora também podemos verificar o KO do CDS com o maior número de *hits* na amostra de tumor *:*

mysql> select * from hsa_ko order by hits desc limit 10;

28. Vamos agora descobrir o número total de *hits* por KO. Para isso, vamos somar os *hits* de todos os CDS de cada KO. Criaremos uma tabela contendo o número de CDS e o total de *hits* para cada KO. Assim podemos mapear os KOs mais representativos na amostra. Use *create table* com a seguinte sintaxe:

mysql> create table ko_hits select ko, count(distinct cds) as total_cds, sum(hits) as total_hits from hsa_ko group by ko;

Note que agora temos uma nova clásula *sum.* Ela tem uma função parecida com a clásula *count*, sendo também obrigatoriamente associada à cláusula *group by.* No entanto, ao invés de efetuar a contagem de registros em cada agrupamento, essa cláusula efetua a soma total dos valores de uma coluna específica dos membros de cada grupo. Nesse caso, a soma total é obtida da coluna *hits.* Dessa forma obtemos o número total de *hits* de cada KO. Na tabela resultante, essa coluna é nomeada *total_hits.*

29. Usando *select* e *order by*, verifique os 10 KOs que possuem o maior número de *hits*.

mysql> select * from ko_hits order by total_hits desc limit 10;

O resultado será:

```
+--------+-----------+------------+
| ko     | total_cds | total_hits |
+--------+-----------+------------+
| K00799 |        13 |       1758 |
| K03105 |         1 |       1386 |
| K02951 |         1 |       1314 |
| K12586 |         1 |       1278 |
| K06635 |         2 |       1190 |
| K02871 |         1 |       1164 |
| K02137 |         1 |       1124 |
| K02984 |         1 |       1046 |
| K03767 |         1 |        921 |
| K05687 |         1 |        908 |
+--------+-----------+------------+
```

Note que o gripo KO com o maior número de *hits* engloba 13 CDS (K00799) e não é o mesmo KO do CDS hsa:6728, contado anteriormente como o CDS com o maior número de *hits* (hsa:6728 é o único CDS do K03105).

30. É importante também colocar descrições associadas aos KOs, para que os identifiquemos mais facilmente. As descrições estão presentes no arquivo tabulado *ko_desc,* localizado em sua pasta *mysql_aula. A*ssim como fizemos com CDS, vamos carregar uma tabela com descrições de KOs. Execute os seguintes comandos em sequencia:

mysql> create table ko_description (ko varchar(11) primary key, description varchar(170), path_desc varchar(150));

mysql> load data local infile '/ko_desc' into table ko_description;

Note que se o arquivo hsa_ko.list não estiver na pasta onde se acessou o MySQL é necessário dar o caminho completo do mesmo.

31. Utilizando *alter table,* como feito anteriormente, vamos adicionar as colunas de descrição do KO na tabela *ko_hits.*

mysql> alter table ko_hits add column ko_desc varchar(400);

32. Utilizando *update,* atualize a tabela *ko_hits* para incluir as descrições dos KOs*:*

mysql> update ko_hits, ko_description set ko_hits.ko_desc = ko_description.description where ko_hits.ko = ko_description.ko;

33. Vamos verificar novamente os 10 KOs que possuem mais *hits.* Dessa vez será possível ver a descrição de cada KO.

mysql> select * from ko_hits order by total_hits desc limit 10;

34. Podemos encontrar um ou mais KOs sabendo apenas uma parte de sua descrição. Para isso, utilizamos o comando *like*. Por exemplo, se quisermos encontrar todos os KOs que possuam o trecho "ATP" em suas descrições, usamos o comando:

mysql> select * from ko_hits where ko_desc like '%ATP%';

Nesse caso, encontramos 65 KOs cujas descrições contém o trecho "ATP". O símbolo "%" atua como um coringa, para encontrar ATP precedido ou sucedido de qualquer caractere, de outra forma a consulta só encontraria um KO cuja descrição fosse somente "ATP", o que não existe. Analogamente, "ATP%" encontraria "ATP-sintase". Poderíamos adicionar *order by* para ordenar o resultado pelo número de *hits*, ou pelo número de CDS, ou mesmo por ordem alfabética.

35. Pense em outras proteínas e repita a consulta

Terminamos com isso nosso tutorial de como usar MySQL para analisar resultados do BLAST. O próximo passo para uma melhor fixação do que foi aprendido é continuar praticando com os comandos e suas variações. Você ficou conhecendo diversos comandos SQL que, usados em conjunto, formam uma poderosa ferramenta de análise. Com os comandos que você aprendeu, pode usar *select* com os mais diversos filtros *where* ou *like* para encontrar resultados bem específicos. Pode também usar *group by*, *count* e *sum* nas mais diversas combinações de colunas. Pode ver resultados ordenados da maneira que desejar e criar tabelas auxiliares.

Existem ainda muitos outros comandos que podem ser encontrados no manual do MySQL [1]. Esse tutorial demonstra como a utilização de bancos de dados torna a análise de dados biológicos muito mais eficiente e produtiva.

Referências

[1] MySQL: http://www.mysql.com
[2] BLAST: http://www.ncbi.nlm.nih.gov/BLAST
[3] Cygwin: http://www.cygwin.com
[4] Formatdb: http://www.ncbi.nlm.nih.gov/staff/tao/URLAPI/formatdb_fastacmd.html