Gestió de dades

Autors: Guerau Fernandez Isern, Joan Colomer Vila, Maria Begoña Hernández Olasagarre, Enrique Blanco García
L'encàrrec i la creació d'aquest recurs d'aprenentatge UOC han estat coordinats pel professor: Josep Jorba Esteve
PID_00298304
Primera edició: setembre 2023

2.11. Agregacions a MongoDB

Resum

Activitats

Exercicis d'autoavaluació

Solucionari

Bibliografia

Introducció

Els fitxers són les unitats lògiques d'informació persistent dins d'un ordinador. Com que manquen d'estructura pròpia, és el programador qui estableix com organitzar la informació en el seu interior (per exemple, fitxers FASTA, XML o JSON).

Mitjançant les ordres apropiades del terminal o línia d'ordres, podem analitzar el contingut dels fitxers d'una manera relativament eficient i còmode. En el mòdul «Introducció als entorns de treball GNU/Linux» hem treballat amb ordres del terminal per gestionar la informació emmagatzemada en fitxers de text que contenen anotacions biològiques. Tanmateix, quan el volum d'informació excedeix certs límits, com és el cas de l'anotació completa d'un genoma, i s'incrementa el nombre de persones involucrades en un projecte de recerca, cal organitzar i estructurar la informació en una base de dades i gestionar-la utilitzant un programa especialitzat, també anomenat Sistema de Gestió de Bases de Dades (SGBD).

Un SGBD és l'eina idònia per administrar eficientment elevades quantitats de registres o dades. La responsabilitat sobre la gestió i els formats interns de les dades correspon al sistema, la qual cosa allibera el propi usuari d'aquestes tasques.

Amb un sistema de gestió de la informació tindrem a la nostra disposició un conjunt d'eines i instruccions que ens permetran extreure nou coneixement de tota aquesta informació.

En aquest mòdul veurem dos models de gestió de les dades, el model relacional basat en el llenguatge SQL, i utilitzarem el SGBD MySQL, i el model no relacional, també anomenat NoSQL, i utilitzarem el SGBD MongoDB basat en col·leccions de documents.

Objectius

1. Conèixer el model entitat-relació per dissenyar bases de dades.
2. Dominar la conversió d'entitats i relacions en taules amb atributs.
3. Crear i administrar una base de dades amb MySQL.
4. Entendre les característiques bàsiques del llenguatge SQL.
5. Crear i poblar amb registres reals les taules SQL.
6. Realitzar consultes SQL a una base de dades relacional.
7. Conèixer la utilitat dels subprogrames emmagatzemats, procediments, funcions i disparadors (Triggers).
8. Iniciar-se amb els fitxers JSON i el SGBD NoSQL MongoDB.
9. Administrar un sistema de gestió de base de dades.

1. Bases de dades relacionals

1.1. Introducció

El model relacional és un model de dades basat en la lògica de predicats i en la teoria de conjunts. La seva idea fonamental és l'ús de relacions. Aquestes relacions podrien considerar-se de forma lògica, com a conjunts de dades anomenades *tuples*. Pensem cada relació com si fos una taula que està composta per registres: cada fila de la taula seria un registre o *tupla*, i columnes, també anomenades *camps*.

Entre els paradigmes actuals de bases de dades, el model relacional està molt estès i s'adapta a la majoria d'entorns bioinformàtics per la seva eficiència i simplicitat.

Un altre avantatge d'aquest paradigma és que existeixen nombroses implementacions *open source* que proporcionen els serveis complets d'un sistema gestor de base de dades relacional amb diferents interfícies gràfiques d'usuari.

Formalment, el paradigma relacional està dividit en tres components bàsics:

- Les taules i les relacions entre aquestes estructuren les dades.
- L'àlgebra relacional opera sobre la informació.
- Un conjunt d'axiomes manté la integritat del sistema.

Una taula modela un element del món real, caracteritzant els seus atributs. Una relació entre dues taules emula les associacions lògiques existents entre dos elements de diferents classes en la realitat, permetent l'accés creuat d'informació.

Per a un univers de dades en particular, l'organització de les taules i les relacions que el conformen reben el nom d'esquema relacional. Una vegada definida aquesta estructura, s'ha de crear una base de dades per ser poblada amb les dades reals (conegudes com a instàncies o registres), sent administrada des d'aquell moment per un sistema de gestió de bases de dades.

Utilitzant l'àlgebra relacional, l'usuari pot realitzar consultes per extreure'n nova informació i actualitzar-la.

Per realitzar un disseny eficient de la base de dades hem de seguir aquestes regles:

- Estimular totes les classes d'informació que desitgem guardar.
- Estructurar de forma lògica la informació en diferents categories.
- Definir els atributs que caracteritzen cada categoria.
- Assignar identificadors suficientment descriptius als atributs.
- Decidir el tipus de dades associat a cada atribut.
- Descompondre cada peça d'informació en la unitat més elemental.
- Seleccionar els atributs que identifiquen de forma única cada categoria.
- Identificar les relacions entre categories.

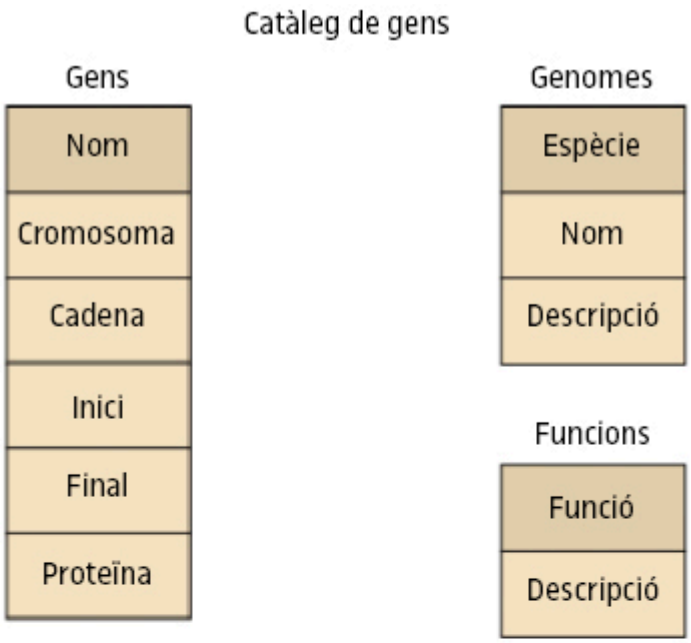
1. Bases de dades relacionals

1.2. El model entitat-relació

Podem estructurar qualsevol realitat en diferents entitats que poden interactuar entre elles seguint determinades regles. Per exemple, el genoma, en tant que part de la realitat biològica d'una cèl·lula, també es pot estructurar en diferents components. A partir d'aquesta organització artificialment construïda, podem modelar la totalitat dels elements que el conformen utilitzant entitats i relacions. Aquestes estructures, contenidors d'informació o dades poden ser digitalitzats en un ordinador per a la seva gestió i anàlisi; en el cas del genoma, anàlisis bioinformàtics.

El model entitat-relació ens ajuda a dissenyar la nostra pròpia base de dades. Una entitat representa una mena d'elements en l'entorn real que desitgem modelar. Una ocurrència és una instància o exemple particular d'una entitat. La connectivitat o participació entre entitats s'ha d'especificar explícitament mitjançant relacions. El nombre d'ocurrències d'una entitat que podran relacionar-se amb instàncies d'una altra entitat serà:

- Un a un (1:1): un element de la primera entitat pot relacionar-se amb un únic element de la segona.
- Un a diversos (1:N): un element de la primera entitat pot relacionar-se amb diversos elements de la segona (però no al contrari).
- Diversos amb diversos (M:N): un element de la primera entitat pot relacionar-se amb diversos elements de la segona (i viceversa).



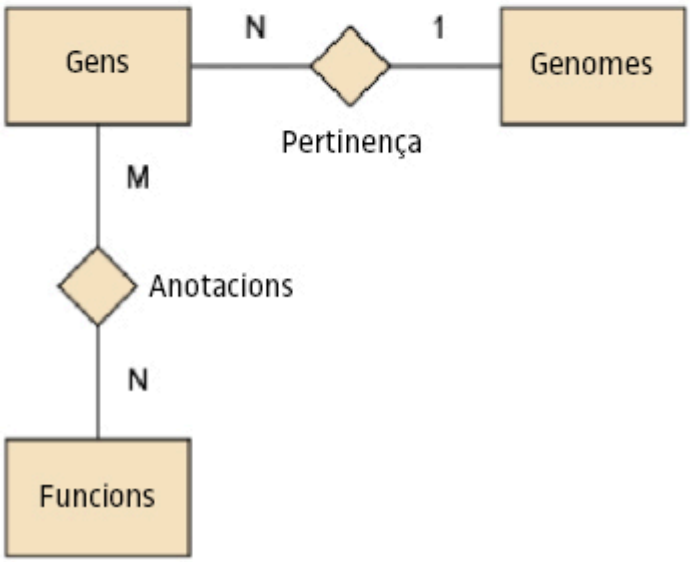


Figura 1. Model entitat-relació d'un catàleg de gens.
Font: elaboració pròpia.

El model conté tres entitats i dues relacions. Gràficament, les entitats es representen utilitzant rectangles, i les relacions, mitjançant línies rectes amb un rombe per indicar la connectivitat.

Aquest esquema basat en entitats i relacions resulta senzill d'emprar posteriorment per construir la base de dades definitiva. Per mostrar les diferents etapes de disseny, procedirem a modelar un escenari molt habitual en entorns de recerca bioinformàtics: la caracterització del catàleg de gens d'un genoma. Podeu observar les diferents entitats amb els seus atributs i les relacions entre entitats que han de formar el nostre model en la figura 1.

Els gens són fragments d'ADN ubicats en una localització precisa del genoma que codifiquen la seqüència d'una proteïna. Aquestes molècules, d'altra banda, exerceixen una funció biològica específica dins de l'organisme (*). Lògicament, cada genoma posseeix el seu propi catàleg de gens. Analitzant aquesta informació prèvia, decidim modelar aquest entorn utilitzant les entitats **gens**, **genomes** i **funcions**, amb els seus propis atributs (mostrats en la figura 2).

Lògicament, les entitats no són objectes aïllats del seu entorn. Per tant, hem de complint les especificacions del nostre problema, unir mitjançant relacions aquelles entitats que estan interconnectades en el món real. En aquest cas, els gens tenen la capacitat de pertànyer a un genoma per desenvolupar una funció concreta en l'organisme. Per satisfer ambdues propietats, definim les relacions binàries **pertinença** i **anotacions**, cadascuna amb una connectivitat diferent (figura 1). Les entitats i relacions que formen aquest model han de ser convertides en taules. Les entitats i els seus atributs passaran a ser taules de la nostra base de dades, però no totes les relacions seran taules, depèn del tipus de connectivitat. És bàsic establir per a cada taula un atribut especial (o una combinació d'atributs) que identifiqui cada instància de forma unívoca. Aquest atribut especial rep el nom de clau primària. És preferible usar un codi característic en lloc d'un nom per facilitar la identificació de qualsevol instància mitjançant la clau primària (per exemple, un codi numèric assignat en funció de l'ordre d'entrada a la taula).

```
GENS (nom, cromosoma, cadena, inici, final, proteina)
GENOMES (especie, nom, descripcio)
FUNCIONS (funcio, descripcio)
```

Figura 2. Entitats convertides en taules.
Hem subratllat la clau primària de cada taula.
Font: elaboració pròpia.

Les dues relacions existents en el nostre esquema s'han de modelar de diferent forma, atès que cadascuna presenta una combinació diferent de possibles ocurrències entre taules. L'associació entre les taules *gens* i *genomes* s'ha de representar amb una relació amb cardinalitat 1:N, ja que un gen només pertany a un genoma, però un genoma conté molts gens. Aquesta relació no necessita una taula nova, pot implementar-se referenciant simplement des d'una taula (gens), que és la part *n* de la relació *pertinença*, la clau primària de l'altra (genomes), que és la part 1 de la relació *pertinença*. Dins de la taula *gens*, aquest atribut rep la denominació de clau forana.

```
GENS (nom, cromosoma, cadena, inici, final, proteina, especie)
GENOMES (especie, nom, descripcio)
```

Figura 3. Relacions (1:N) convertides en claus foranes.
Indiquem amb un subratllat superior la clau forana de cada taula.
Font: elaboració pròpia.

La relació *anotacions* entre les taules *gens* i *funcions* té una connectivitat M:N, ja que un gen pot posseir diverses anotacions, però una anotació també pot ser compartida per diversos gens. Per a la seva correcta implementació, hi introduirem una nova taula anomenada *anotacions*. Aquesta posseirà, com a clau primària, la combinació de les claus primàries de cada taula original.

Atès que ambdues claus primàries per separat posseeixen totes les propietats necessàries, el dissenyador garanteix amb aquesta mesura que cada instància d'aquesta nova taula estarà dotada d'un identificador únic, que estarà format pel nom del gen juntament amb el nom de la funció en particular perquè sigui un identificador de la instància únic que no es pugui repetir.

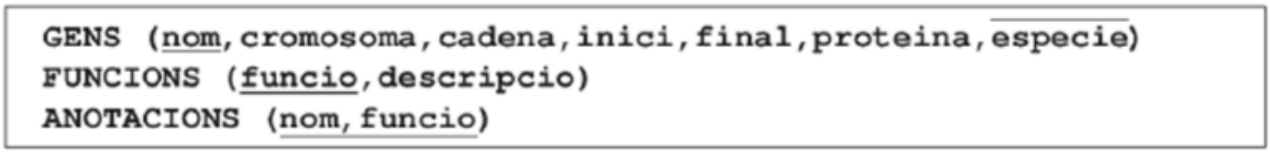


Figura 4. Relacions (M:N) convertides en taules.
Font: elaboració pròpia.

Les relacions amb cardinalitat M:N exportades des del model entitat-relació estan caracteritzades també pels seus propis atributs. Així, és possible afegir un camp per guardar l'origen de cada anotació (per exemple, computacional, experimental o font bibliogràfica). Podem extreure aquesta informació a partir de les dades recuperades del sistema d'anotació automàtica utilitzat per poblar d'exemples el nostre catàleg de gens:

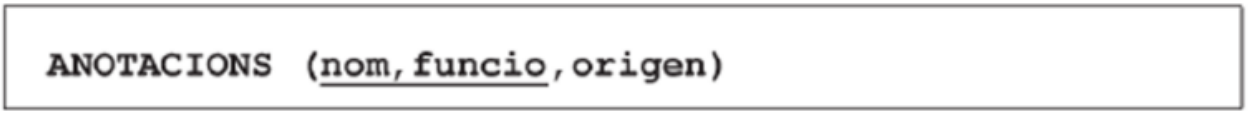


Figura 5. Atributs en relacions convertides en taules.
Font: elaboració pròpia.

La selecció de les claus idònies resulta essencial dins del disseny d'una base de dades relacional. De fet, la integritat d'un model relacional ha de complir dos requisits fonamentals relacionats amb la gestió d'aquestes:

1. La clau primària no ha de contenir un valor indefinit o nul i el valor ha de ser únic.

2. La clau forana ha de fer referència a una clau primària d'una altra taula.

1. Bases de dades relacionals

1.3. El llenguatge SQL i el SGBD MySQL

L'SQL (en anglès, *Structured Query Language*, 'llenguatge de consultes estructurades') és el llenguatge d'accés a les bases de dades relacionals més estès. Amb aquest sistema, el client especifica les instruccions per crear, dotar de contingut, modificar o eliminar les taules de la base de dades, i realitzar les consultes.

Per treballar amb el llenguatge SQL és necessari disposar d'un sistema gestor de bases de dades (SGBD), una aplicació informàtica normalment basada en el model client-servidor per administrar bases de dades relacionals.

Existeixen diferents SGBD, com Oracle, PostgreSQ... En aquest mòdul farem servir el SGBD MySQL. A Linux, el servidor MySQL funciona en segon pla, sense interferir en la planificació de processos del sistema. D'aquesta manera, quan l'usuari desitja utilitzar la base de dades, s'ha de connectar amb l'aplicació gestora mitjançant un programa client, a través d'un entorn gràfic o des del propi terminal amb l'intèrpret d'ordres de SQL, denominat **mysql**.

SQL va ser comercialitzat per IBM el 1981. MySQL és un gestor distribuït per Oracle sota llicència GNU o comercial.

1. Bases de dades relacionals

1.4. Començar a treballar amb MySQL

A la màquina virtual que us proporcionem hi ha instal·lat un SGBD MySQL. En iniciar la màquina virtual també s'inicia el servidor MySQL.

El sistema client-servidor permet accedir a un únic servidor des de diversos clients. Per defecte, el client del sistema és un terminal o línia d'ordres, però es pot accedir també al servidor des d'un client amb interfície gràfica d'usuari (GUI).

A la màquina virtual hi ha instal·lada la GUI **MySQL Workbench**, però és possible accedir al servidor MySQL des de molts clients amb diferents GUI.

Per començar a treballar amb MySQL utilitzarem el terminal de Linux executant l'ordre mysql.

Una vegada establerta la connexió al servidor, el programa client roman sempre a l'espera de la introducció d'una nova ordre SQL per part de l'usuari.

És important ser curós amb la sintaxi de les ordres, i finalitzar cada instrucció amb el símbol « ; ».

Per il·lustrar el funcionament del joc d'instruccions de SQL mostrat a la taula 1 sobre el model relacional anterior (vegeu figura 4), implementarem una base de dades que denominarem **cataleg**.

Taula 1. Manual de referència d'ordres de MySQL.

Ordre	Descripció
CREATE USER	Donar d'alta un nou usuari
DROP USER	Donar de baixa un usuari existent
ALTER USER	Modificar el compte d'un usuari
GRANT	Autoritzar un usuari sobre una base de dades
REVOKE	Revocar les autoritzacions d'un usuari
SHOW GRANTS	Mostrar les autoritzacions d'un usuari
CREATE DATABASE	Crear una nova base de dades
DROP DATABASE	Eliminar una base de dades existent
USE DATABASE	Accedir a una base de dades existent
SHOW DATABASES	Mostrar la llista de les bases de dades
CREATE TABLE	Crear una nova taula
DROP TABLE	Eliminar una taula existent
SHOW TABLES	Mostrar la llista de les taules
DESCRIBE	Mostrar els atributs d'una taula
LOAD DATA	Poblar una taula amb un fitxer de registres
INSERT	Poblar una taula amb un registre
UPDATE	Actualitzar un registre de la taula
DELETE	Esborrar un registre de la taula
SELECT	Realitzar una consulta sobre una o més taules

`Help`	Mostrar ajuda sobre una ordre del sistema
`Pager`	Mostrar llistats pàgina a pàgina amb un altre programa
`Source`	Executar un *script* d'ordres de SQL
`Status`	Mostrar informació sobre l'estat del sistema
`System`	Executar una ordre del terminal
`Warnings`	Mostrar avisos del sistema
`Quit`	Sortir del gestor MySQL
`Exit`	Sortir del gestor MySQL
`Mysql`	Ordre del terminal per invocar el gestor MySQL
`Mysqldump`	Ordre del terminal per al *backup* d'una base de dades

Font: elaboració pròpia.

El SGBD MySQL permet gestionar múltiples bases de dades, implementant un sistema de seguretat basat en autoritzacions. Un cop instal·lat el servidor al nostre entorn Linux, es crea inicialment un usuari amb el rol d'administrador (*) (en anglès, *root*) amb tots els permisos. Per regla general, només l'usuari amb aquest perfil posseeix els permisos suficients per gestionar el conjunt d'usuaris i bases de dades en la seva totalitat. D'aquesta manera, l'administrador garantirà l'accés a una determinada base de dades exclusivament a un grup d'usuaris establert prèviament. Malgrat que es pot optar per treballar directament com a administrador, sempre és recomanable que configurem un nou compte al nostre nom com a usuari convencional per autoritzar-lo posteriorment a treballar amb una base de dades en concret.

En definitiva, el protocol que hem de seguir per dur a terme la nostra feina amb el gestor de bases de dades consisteix en dues fases:

1. Invoquem l'intèrpret de SQL com a administrador, realitzem les següents tasques i abandonem el programa:
    1. Creem un nou usuari al nostre nom.
    2. Creem una nova base de dades.
    3. Autoritzem el nou usuari a treballar amb aquesta base de dades.
2. Accedim a l'intèrpret de SQL amb el nostre propi usuari i procedim a interactuar amb la nova base de dades:
    1. Especifiquem que anem a treballar amb aquesta base de dades.
    2. Creem noves taules (buides) dins de la base de dades.
    3. Inserim nous registres en aquestes taules. També podem modificar-los i eliminar-los.
    4. Realitzem consultes sobre els registres de les taules per obtenir la informació desitjada.

Per començar a treballar primer hem d'executar el programa **mysql** des del nostre terminal de Linux emprant l'usuari *root*:

```
% mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or g.
Your MySQL connection id is 5
Server version: 5.7.17-0ubuntu0.16.04.1 (Ubuntu)
Copyright (c) 2000, 2016, Oracle and/or its affiliates. All
rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or
its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or 'h' for help. Type 'c' to clear the current
input statement.

mysql>
```

Figura 6. Iniciar el gestor MySQL com a administrador.
Font: elaboració pròpia.

L'ordre CREATE USER permet a l'usuari ***root*** donar d'alta un nou usuari en el nostre sistema. És recomanable assignar una contrasenya a cada nou usuari del nostre sistema:

```
CREATE USER usuari

IDENTIFIED BY password;
```

Figura 7. Sintaxi de l'ordre CREATE USER.
Font: elaboració pròpia.

Ara, actuant com a administradors, crearem un nou usuari anomenat **eblanco**. Per indicar que aquest usuari treballarà localment des de la nostra màquina, que actua de servidor, emprarem el terme **local**.

En alguns exemples, perquè siguin més comprensibles, estructurarem les instruccions de SQL en diverses línies. Després d'introduir cada línia de l'ordre pressionant la tecla «Enter», l'intèrpret de MySQL inserirà els símbols -> per denotar que encara no hem acabat d'introduir l'ordre completa. MySQL no procedirà a executar l'ordre fins a reconèixer el caràcter «;».

Us animem a reproduir al vostre ordinador les ordres de SQL presentades en aquesta unitat.

```
mysql> CREATE USER 'eblanco'@'localhost'
    -> IDENTIFIED BY '123456';

Query OK, 0 rows affected (0,35 sec)
```

Figura 8. Crear un nou usuari amb l'administrador.
Font: elaboració pròpia.

Existeixen diverses ordres per gestionar el conjunt de bases de dades del sistema (consulteu taula 1). En aquest moment hem de procedir a crear la base de dades on treballarà el nou usuari amb l'ordre CREATE DATABASE:

```
CREATE DATABASE basededades;
```

Figura 9. Sintaxi de l'ordre CREATE DATABASE.
Font: elaboració pròpia.

A continuació, creem la base de dades **cataleg** i posteriorment emprem l'ordre SHOW DATABASES per obtenir el llistat de bases de dades existents. Podem comprovar que la nova base de dades ha estat creada correctament:

```
mysql> CREATE DATABASE cataleg;

Query OK, 1 row affected (0,00 sec)

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| cataleg            |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0,10 sec)
```

Figura 10. Creació de la base de dades *cataleg*.
En finalitzar l'execució d'una ordre, l'intèrpret mostra per pantalla el nombre d'elements dels resultats (en anglès, *rows*).
Font: elaboració pròpia.

L'administrador concedeix permisos sobre les operacions que un determinat grup d'usuaris pot realitzar sobre la base de dades. L'ordre GRANT permet autoritzar l'accés d'un usuari a una base de dades en particular:

```
GRANT operacions ON basededades

TO usuari IDENTIFIED BY password;
```

Figura 11. Sintaxi de l'ordre GRANT.
Font: elaboració pròpia.

Per tancar la feina de l'administrador, hem de garantir l'accés a la nova base de dades **cataleg** al nostre usuari **eblanco** emprant l'ordre GRANT. Amb la clàusula **ALL**, l'administrador concedeix el conjunt complet de permisos a aquest usuari, encara que exclusivament sobre aquesta base de dades. Si volem eliminar privilegis a un usuari hem d'utilitzar l'ordre REVOKE.

Finalment, per abandonar l'execució del programa **mysql** com a administradors, podem emprar les ordres `quit` o `exit`.

```
mysql> GRANT ALL ON cataleg.* TO 'eblanco'@'localhost';

Query OK, 0 rows affected (0,21 sec)

mysql> quit;
```

Figura 12. Autoritzar l'accés a un nou usuari.
Font: elaboració pròpia.

A partir d'aquest instant, a l'hora d'executar el programa **mysql** treballarem sobre la nostra base de dades amb el nou usuari **eblanco**. En primer lloc, hem d'accedir al gestor de MySQL emprant el nom d'usuari i la contrasenya que hem creat.

```
% mysql -u eblanco -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or g.
Your MySQL connection id is 8
Server version: 5.7.17-0ubuntu0.16.04.1 (Ubuntu)
Copyright (c) 2000, 2016, Oracle and/or its affiliates. All
rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or
its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or 'h' for help. Type 'c' to clear the current
input statement.

mysql>
```

Figura 13. Iniciar el gestor MySQL com un usuari convencional.
Font: elaboració pròpia.

Com que és la primera vegada que hi accedim amb aquest usuari, obtindrem informació sobre el llistat de les bases de dades accessibles utilitzant l'ordre SHOW DATABASES.

Com mostrem a continuació, en la figura 14, el nostre nou usuari pot treballar amb dues bases de dades: **cataleg** i una segona base de dades (**information_schema**) que conté informació interna sobre la configuració del sistema.

Amb l'ordre SHOW GRANTS podem veure també les autoritzacions que l'administrador ha concedit a aquest usuari.

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| catalogo           |
+--------------------+
2 rows in set (0,00 sec)

mysql> SHOW GRANTS;
+-----------------------------------------------------------------+
| Grants for eblanco@localhost                                    |
+-----------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'eblanco'@'localhost'
|
| GRANT ALL PRIVILEGES ON `catalogo`.* TO 'eblanco'@'localhost'
|
+-----------------------------------------------------------------+
2 rows in set (0,00 sec)
```

Figura 14. Conèixer les bases de dades disponibles.
Font: elaboració pròpia.

Qualsevol usuari, una vegada dins del sistema, ha d'especificar el nom de la base de dades que ha d'utilitzar abans de començar a realitzar operacions sobre ella. Per indicar el nom de la base de dades que seleccionarem, farem servir l'ordre **USE**:

```
USE basededades;
```

Figura 15. Sintaxi de l'ordre USE.
Font: elaboració pròpia.

Com que desitgem treballar amb la base de dades **cataleg**, procedim a declarar aquest fet en l'intèrpret de MySQL. Un cop aquesta instrucció ha estat executada amb èxit, ja estem en disposició de crear les taules que serveixen com a suport de les entitats i les relacions dissenyades amb anterioritat per poblar-les posteriorment amb nous registres.

```
mysql> USE cataleg;

Database changed
```

Figura 16. Seleccionar la base de dades per treballar.
Font: elaboració pròpia.

1. Bases de dades relacionals

1.5. Creació de taules i restriccions

Una base de dades està formada per un conjunt de taules que ens permetran estructurar la informació que percebem en un escenari concret del món real. Cada taula emmagatzemarà en forma de registres la sèrie d'exemples de cada classe d'entitats o relacions entre entitats especificades prèviament. Per crear una taula de registres buida amb l'ordre CREATE TABLE cal primer declarar els seus atributs (consulteu la figura 17).

```
CREATE TABLE nom
(camp1 tipus1 [NOT NULL, AUTO_INCREMENT],
 camp2 tipus2 [NOT NULL, AUTO_INCREMENT],
 ...
 campN tipusN [NOT NULL, AUTO_INCREMENT],
 PRIMARY KEY(campx,campy,...),
 [FOREIGN KEY(campx,campy,...)
 REFERENCES taula(campa,campb,...)]);
```

Figura 17. Sintaxi de l'ordre CREATE TABLE.
Els elements opcionals es mostren entre claudàtors.
Font: elaboració pròpia.

A l'hora de definir la classe d'informació que emmagatzemarem en cada atribut, MySQL proporciona una gran varietat de tipus numèrics i alfanumèrics bàsics (taula 2). L'espai de memòria requerit per emmagatzemar cada variable depèn de la precisió especificada en cada cas. Els tipus DATE i TIME resulten especialment útils per portar el registre de les nostres activitats en el temps. L'usuari pot declarar, a més, variables del tipus objecte (en anglès, *Binary Large Objects* o BLOB) per emmagatzemar fitxers de text, documents en format PDF o fins i tot imatges dins d'alguna taula de la base de dades.

Taula 2. Tipus de dades en MySQL.

Tipus genèric	Tipus MySQL
Sencer	TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT
Decimal	DECIMAL, FLOAT, DOUBLE/REAL
Text	CHAR, VARCHAR, TINYTEXT, TEXT
Objectes	BLOB, MEDIUMBLOB, LONGBLOB
Temps	DATE, TIME

Font: elaboració pròpia.

Juntament amb la declaració dels atributs, per crear una taula hem d'especificar quin atribut o combinació d'atributs serà la clau primària, identificant de forma unívoca cada instància (figura 18). En cas d'existir, les claus foranes per referenciar els atributs d'altres taules també s'han d'indicar explícitament.

El dissenyador pot activar dos controls interns sobre el valor d'un atribut en el moment de registrar noves instàncies a la base de dades. En primer lloc, és possible rebutjar aquells registres que no posseeixin un valor definit per a un atribut concret. Aquesta circumstància s'especifica amb la construcció NOT NULL, just després de la declaració de tipus. NOT NULL seria una

restricció de camp obligatori, no accepta valors nuls. En cas contrari, el sistema assignarà per defecte el valor NULL a aquest camp i acceptarà valors nuls. Aquest requeriment s'ha de satisfer inexcusablement en aquells atributs que pertanyen a la clau primària. En segon lloc, per als identificadors numèrics associats a cada instància, el propi sistema pot encarregar-se de gestionar un comptador automàtic de valors mitjançant la construcció AUTO_INCREMENT.

Tornant novament a la nostra base de dades **cataleg** (figura 16), ens trobem ara en disposició de crear les taules del catàleg de gens especificades formalment a les figures 18, 19, 20 i 21. Hem d'escollir adequadament els tipus de dades per a cada atribut o camp, segons el seu contingut, definint clarament quines són les claus primàries i foranes. L'usuari pot començar creant les taules més elementals, és a dir, aquelles que no posseeixen claus foranes (genomes i funcions). Observeu com declarem la clau primària i ens assegurem que cap instància pot donar-se d'alta a la base de dades amb un valor nul per a les claus primàries, **espècie** i **funció**. Per verificar que el procés de creació ha funcionat correctament, l'usuari pot consultar la base de dades sobre les taules existents amb l'ordre SHOW TABLES.

```
mysql> SHOW TABLES;

Empty set (0,00 sec)

mysql> CREATE TABLE genomes
    -> (especie     VARCHAR(100) NOT NULL,
    ->  nom         VARCHAR(100),
    ->  descripcio  TEXT,
    ->  PRIMARY KEY (especie));

Query OK, 0 rows affected (0,28 sec)

mysql> CREATE TABLE funcions
    -> (funcio      VARCHAR(20) NOT NULL,
    ->  descripcio  VARCHAR(100),
    ->  PRIMARY KEY (funcio));

Query OK, 0 rows affected (0,03 sec)

mysql> SHOW TABLES;

+-------------------+
| Tables_in_cataleg |
+-------------------+
| funcions          |
| genomes           |
+-------------------+
2 rows in set (0,00 sec)
```

Figura 18. Crear les taules *genomes* i *funcions* en la nostra base de dades *cataleg*.
Font: elaboració pròpia.

És possible revisar la definició d'una taula amb la instrucció DESCRIBE:

```
mysql> DESCRIBE genomes;

+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| especie    | varchar(100) | NO   | PRI | NULL    |       |
| nom        | varchar(100) | YES  |     | NULL    |       |
| descripcio | text        | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
3 rows in set (0,01 sec)
```

Figura 19. Mostra de la descripció d'una taula del nostre catàleg.
Font: elaboració pròpia.

A continuació, per crear la taula **gens**, a més de la clau primària, indiquem un camp o atribut que és la clau forana (que ha d'apuntar o fer referència a la clau primària de la taula *genomes*):

```
mysql> CREATE TABLE gens
    -> (nom        VARCHAR(20) NOT NULL,
    ->  cromosoma  VARCHAR(5),
    ->  cadena     VARCHAR(1),
    ->  inici      INT,
    ->  final      INT,
    ->  proteina   VARCHAR(20),
    ->  especie    VARCHAR(100),
    ->  PRIMARY KEY (nom),
    ->  FOREIGN KEY (especie)
    ->  REFERENCES genomes(especie));

Query OK, 0 rows affected (0,06 sec)

mysql> DESCRIBE gens;

+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| nom      | varchar(20)  | NO   | PRI | NULL    |       |
| cromosoma | varchar(5)  | YES  |     | NULL    |       |
| cadena   | varchar(1)   | YES  |     | NULL    |       |
| inici    | int(11)      | YES  |     | NULL    |       |
| final    | int(11)      | YES  |     | NULL    |       |
| proteina | varchar(20)  | YES  |     | NULL    |       |
| especie  | varchar(100) | YES  | MUL | NULL    |       |
+----------+-------------+------+-----+---------+-------+
7 rows in set (0,00 sec)
```

Figura 20. Creació de la taula *gens* a la nostra base de dades.
Font: elaboració pròpia.

Finalment, creem la taula **anotacions** per relacionar els gens amb les anotacions funcionals. Juntament amb els dos valors que identifiquen cada registre (**gen** i **funció**), afegirem un atribut per registrar l'origen de la informació:

```
mysql> CREATE TABLE anotacions
    -> (nom          VARCHAR(20) NOT NULL,
    ->  funcio      VARCHAR(20) NOT NULL,
    ->  origen       VARCHAR(20),
    ->  PRIMARY KEY (nom,funcio),
    ->  FOREIGN KEY (nom)
    ->  REFERENCES  gens(nom),
    ->  FOREIGN KEY (funcio)
    ->  REFERENCES  funcions(funcio));

Query OK, 0 rows affected (0,11 sec)

mysql> DESCRIBE anotacions;

+--------+------------+------+-----+---------+-------+
| Field  | Type       | Null | Key | Default | Extra |
+--------+------------+------+-----+---------+-------+
| nom    | varchar(20) | NO   | PRI | NULL    |       |
| funcio | varchar(20) | NO   | PRI | NULL    |       |
| origen | varchar(20) | YES  |     | NULL    |       |
+--------+------------+------+-----+---------+-------+
3 rows in set (0,01 sec)
```

Figura 21. Creació de la taula *anotacions* a la nostra base de dades *cataleg*.
Els dos components de la clau primària de la taula *anotacions* s'han de declarar com a claus foranes amb origen en les taules *gens* i *funcions*.
Font: elaboració pròpia.

És possible definir altres restriccions als camps de les taules amb l'ordre CHECK.

Per exemple, podem indicar que un camp numèric com el camp **inci** de la taula **gens** que és tipus numèric només accepti valors positius `CHECK (inici >0)`, o que un camp de tipus cadena de caràcters només accepti determinats valors; per exemple, perquè el camp *hebra* de la taula *gens* només accepti els caràcters «+» o «-», podem fer `hebra ENUM('+','-')`.

Un cop creada la taula podem modificar-la amb la instrucció **ALTER TABLE**.

Per exemple, si volem eliminar el camp origen de la taula **anotacions** escrivim

```
ALTER TABLE anotacions DROP COLUMN origen;
```

i si volem tornar a afegir-hi el mateix camp escrivim

```
ALTER TABLE anotacions ADD origen VARCHAR(20);
```

Durant el temps de vida d'una base de dades és freqüent que n'haguem d'actualitzar el contingut. En determinats casos, això pot implicar l'eliminació completa d'usuaris, de taules o, fins i tot, de la pròpia base de dades. Per implementar aquests serveis, MySQL posseeix la família d'ordres DROP.

```
DROP DATABASE basededades;

DROP USER usuari;

DROP TABLE taula;
```

Figura 22. Sintaxi de l'ordre DROP.
Font: elaboració pròpia.

1. Bases de dades relacionals

1.6. Inserir i manipular dades a les taules

Un cop l'esquema relacional d'entitats prèviament dissenyat està estructurat sobre MySQL mitjançant taules, és el moment de dotar de contingut cada taula.

MySQL disposa de dues formes d'inserir nous registres a les taules de la base de dades:

1. Càrrega simultània de múltiples registres des d'un fitxer de text.

2. Inserció individual de cada nou registre de forma manual.

Per importar una quantitat elevada de registres (*) és possible utilitzar l'ordre LOAD DATA. Per invocar aquesta ordre hem d'especificar el nom del fitxer de text que alberga la informació dels registres juntament amb el nom de la taula on han de ser donats d'alta. Cal disposar d'accés a un fitxer de text tabulat, on cada fila representa un nou registre i cada columna alberga el valor d'un atribut (especificat en el mateix ordre que a la taula).

```
LOAD DATA LOCAL INFILE fitxer.txt INTO TABLE taula;
```

> En determinats entorns d'UNIX cal activar específicament l'opció –local-infile a l'hora d'invocar el programa mysql. Aquesta opció, no obstant, està activada per defecte habitualment en la majoria de distribucions de Linux.

Podem procedir a introduir les primeres dades en el nostre catàleg de gens. És recomanable començar per les taules més elementals, aquelles que no posseeixen claus foranes. En el nostre cas, les taules **genomes** i **funcions** s'ajusten perfectament a aquesta descripció. Per exemple, per poblar la taula **genomes** editarem el següent fitxer, **genomes.txt**, des del nostre terminal d'UNIX:

```
D. melanogaster Mosca de la fruita   Tambe anomenada del vinagre
H. sapiens      Home                 La nostra propia especie
M. musculus     Ratoli               Un altre organisme model
```

Per procedir a la càrrega d'aquestes dades a la taula *genomes*, l'usuari ha d'introduir la següent ordre des de l'intèrpret de MySQL:

```
mysql> LOAD DATA LOCAL INFILE 'genomes.txt' INTO TABLE genomes;

Query OK, 3 rows affected (0,08 sec)
Records:  3  Deleted:  0  Skipped:  0  Warnings:  0
```

Presentem el contingut del fitxer **funcions.txt,** que emprarem per poblar la taula *funcions* amb tres nous registres:

```
GO:0003700          Factor de transcripcio
GO:0006338          Remodelatge de cromatina
GO:0007254          Via JNK
```

Ara emprem el fitxer **funcions.txt** per poblar la corresponent taula:

```
mysql> LOAD DATA LOCAL INFILE 'funcions.txt' INTO TABLE funcions;
Query OK,  3 rows affected (0,01 sec)
Records:  3  Deleted:  0  Skipped:  0  Warnings:  0
```

Un cop hem poblat les taules **genomes** i **funcions** amb diverses instàncies d'espècies i funcions biològiques, respectivament, és el moment d'editar el fitxer **gens.txt** per donar d'alta nous gens a la taula *gens*:

```
MYC    chr8   + 128748314 128753678 NP_002458 H.sapiens
```

```
HNF1A chr12 + 121416548 121440312 NP_000536 H.sapiens
cbt   chr2L -    476437    479046 NP_722636 D.melanogaster
ash2  chr3R +. 20477248  20479098 NP_733023 D.melanogaster
```

I ara procedim a realitzar la càrrega amb l'ordre **LOAD DATA**:

```
mysql> LOAD DATA LOCAL INFILE 'gens.txt' INTO TABLE gens;

Query OK,  4 rows affected (0,02 sec)
Records:  4  Deleted:  0  Skipped:  0  Warnings:  0
```

Abans de procedir a realitzar les primeres consultes, editarem el fitxer de text anotacions.txt per assignar funcions als gens que hem registrat en les ordres prèvies.

```
MYC    GO:0003700  Experimental
MYC    GO:0006338  Literatura
HNF1A  GO:0003700  Experimental
cbt    GO:0003700  Experimental
cbt    GO:0007254  Experimental
ash2   GO:0006338  Experimental
ash2   GO:0003700  Computacional
```

> Observeu que els gens poden posseir més d'una anotació funcional. D'altra banda, la mateixa funció biològica pot ser exercida per gens diferents. Però els dos valors junts formen una clau única.

Estem en condicions de poblar la nostra última taula, *anotacions*:

```
mysql> LOAD DATA LOCAL INFILE 'anotacions.txt'
    -> INTO TABLE anotacions;

Query OK,  7 rows affected (0,08 sec)
Records:  7  Deleted:  0  Skipped:  0  Warnings:  0
```

La càrrega de dades des d'un fitxer de text a les taules és extremadament útil. No obstant això, en determinats casos necessitem donar d'alta un nou registre de forma aïllada, però l'edició d'un fitxer de text únicament amb aquest objectiu és menys eficient.

En aquests casos, l'ordre **INSERT** és més adequada, atès que implementa aquesta funcionalitat en el gestor MySQL de bases de dades. L'usuari ha d'especificar en el mateix ordre tant la llista d'atributs del nou registre com els seus corresponents valors. La resta d'atributs no inclosos en la relació anterior prendran el valor **NULL** (excepte per a aquells on està expressament prohibida aquesta circumstància durant la creació de la taula, camps obligatoris):

```
INSERT INTO taula (camp1, camp2,...,campN)
VALUES (valor1,valor2,...,valorN);
```

Per regla general, però, un registre conté tots els camps declarats per a una taula. Per tant, respectant l'ordre dels camps a la taula, podem ometre la relació completa dels atributs:

```
INSERT INTO taula
VALUES (valor1,valor2,...,valorN);
```

A tall d'exemple, mostrem a continuació la seqüència d'ordres d'inserció equivalent a la càrrega simultània de les funcions executada anteriorment.

Las cadenes de text s'han d'introduir utilitzant sempre cometes simples, mentre que els valors numèrics no necessiten cap format addicional.

```
mysql> INSERT INTO funcions
    -> VALUES ('GO:0003700',
    ->         'Factor de transcripcio');

mysql> INSERT INTO funcions
    -> VALUES ('GO:0006338',
    ->         'Remodelatge de cromatina');

mysql> INSERT INTO funcions
    -> VALUES ('GO:0007254',
    ->         'Via JNK');
```

En el cas d'intentar donar d'alta un registre la clau primària del qual ja existeix, el sistema ens advertirà de l'error, avortant aquesta operació. Una taula no pot tenir una clau primària repetida

Un cop tenim les dades introduïdes a les taules podem modificar-les amb la instrucció UPDATE o eliminar registres amb la instrucció DELETE.

Per exemple, si volem modificar el valor «**Factor de transcripció**», situat en el camp del registre o fila amb clau primària GO:0003700 de la taula, i volem que el nou valor sigui «**Transcription factor**», escriurem la instrucció següent:

```
UPDATE funcions
SET descripcion ='Transcription factor'
WHERE funcion = 'GO:0003700';
```

Per eliminar únicament alguns registres d'una determinada taula podem utilitzar l'ordre DELETE juntament amb la clàusula WHERE. D'aquesta manera, seleccionarem amb precisió els registres que han de ser donats de baixa.

Si l'usuari desitja eliminar tots els registres d'una taula, conservant l'estructura d'aquesta (per reutilitzar-la en el futur), n'hi ha prou amb ometre la condició:

```
DELETE FROM taula WHERE condicions;
DELETE FROM taula;
```

Si volem eliminar el registre de la taula **funcions** amb clau primària **GO:0007254** escriurem la instrucció següent:

```
DELETE FROM funcions WHERE funcion = 'GO:0007254';
```

Si intentem eliminar un registre que està referenciat per una clau forana d'una altra taula el sistema ho impedirà i saltarà un error, tret que a la clau forana li indiquem l'opció **ON DELETE CASCADE**, que permet eliminar en cascada el registre que desitgem eliminar i els registres de l'altra taula que estan referenciats.

1. Bases de dades relacionals

1.7. Consultes bàsiques a les taules

Les bases de dades són eines excepcionalment útils per consultar informació i extreure nou coneixement. En aquest sentit, els esquemes entitat-relació no són una excepció. Més aviat al contrari, utilitzant l'àlgebra relacional és possible consultar el contingut de les taules de múltiples formes. Bàsicament, les consultes SQL (en anglès, *queries*) consisteixen en filtres que delimiten el segment del conjunt complet dels registres d'una o més taules en el qual estem interessats, per mostrar després el valor dels seus atributs.

La instrucció **SELECT** implementa el procés de consulta sobre les taules, mostrant els atributs indicats per a aquells registres que compleixen una determinada condició.

> Podeu trobar informació sobre l'ús del terminal per dur a terme operacions similars sobre fitxers de text al mòdul «L'entorn de treball UNIX».

Anem a explorar en els pròxims anys com enriquir les nostres consultes, utilitzant per a això el nostre catàleg de gens, que està emmagatzemat en la base de dades **cataleg**. Abans de fer consultes més elaborades, mostrem a continuació la forma més senzilla de realitzar una consulta.

```
SELECT camp1,camp2,...,campn FROM taula;
```

Figura 23. Sintaxi bàsica de l'ordre SELECT.
Font: elaboració pròpia.

La consulta més habitual consisteix a mostrar el contingut complet d'una taula. El caràcter *, precisament, indica que desitgem visualitzar el llistat íntegre dels valors de tots els atributs per al subconjunt de registres seleccionats. Per posar en pràctica aquesta ordre sobre el nostre catàleg, seleccionem tots els valors de cada instància guardada a la taula *gens*:

```
mysql> SELECT * FROM gens;
+--------+-----------+--------+-----------+-----------+-----------+-----------------+
|  nom   | cromosoma | cadena |   inici   |   final   |  proteina |     especie     |
+--------+-----------+--------+-----------+-----------+-----------+-----------------+
| ash2   | chr3R     |   +    |  20477248 |  20479098 | NP_733023 | D. melanogaster |
| cbt    | chr2L     |   -    |    476437 |    479046 | NP_722636 | D. melanogaster |
| HNF1A  | chr12     |   +    | 121416548 | 121440312 | NP_000536 | H. sapiens      |
| MYC    | chr8      |   +    | 128748314 | 128753678 | NP_002458 | H. sapiens      |
+--------+-----------+--------+-----------+-----------+-----------+-----------------+
4 rows in set (0,00 sec)
```

Figura 24. Mostrant el contingut íntegre de la taula *gens*.
Font: elaboració pròpia.

Per evitar l'excés d'informació, l'usuari pot seleccionar els atributs o camps dels registres d'una taula que desitja veure per pantalla. Simplement substituint en la pregunta el símbol * per un llistat d'atributs, separats per comes, podem delimitar la vista dels registres, en aquest cas gens que obtenim com a resultat:

```
mysql> SELECT nom,especie FROM gens;

+--------+-----------------+
|   nom  |     especie     |
+--------+-----------------+
| ash2   | D. melanogaster |
| cbt    | D. melanogaster |
| HNF1A  | H. sapiens      |
| MYC    | H. sapiens      |
+--------+-----------------+
4 rows in set (0,00 sec)
```

Figura 25. Mostrant els valors d'alguns atributs de la taula *gens*.
Font: elaboració pròpia.

El recompte del nombre de registres que compleix una condició concreta és una de les consultes més freqüents en SQL. En aquest cas és suficient amb afegir la funció COUNT a la consulta que estem realitzant per comptabilitzar el nombre de línies de la sortida:

```
mysql> SELECT COUNT(*) FROM gens;

+----------+
| COUNT(*) |
+----------+
|        4 |
+----------+
1 row in set (0,00 sec)
```

Figura 26. Comptant tots els registres de la taula *gens*.
Font: elaboració pròpia.

La funció DISTINCT elimina els resultats duplicats. Per exemple, si desitgem comptar el nombre d'organismes a la nostra taula *gens*, podem combinar les funcions COUNT i DISTINCT sobre l'atribut *especie* de la manera següent:

```
mysql> SELECT especie FROM gens;

+----------------+
| especie        |
+----------------+
| D. melanogaster |
| D. melanogaster |
| H. sapiens      |
| H. sapiens      |
+----------------+
4 rows in set (0,00 sec)

mysql> SELECT DISTINCT especie FROM gens;

+----------------+
| especie        |
+----------------+
| D. melanogaster |
| H. sapiens      |
+----------------+
2 rows in set (0,07 sec)

mysql> SELECT COUNT(DISTINCT especie) FROM gens;

+------------------------+
| COUNT(DISTINCT especie) |
+------------------------+
|                      2 |
+------------------------+
1 row in set (0,07 sec)
```

Figura 27. Comptant registres únics d'una taula.
Font: elaboració pròpia.

L'ordre ORDER BY ordena la llista de resultats produïda per una ordre SELECT, de forma ascendent o descendent (segons si hi afegim la clàusula ASC o DESC, respectivament). En la propera figura ordenem els gens per la seva posició en cada cromosoma o pel seu nom, de diferents maneres:

```
mysql> SELECT nom,cromosoma,inici FROM gens ORDER BY inici;

+--------+-----------+-----------+
|  nom   | cromosoma |   inici   |
+--------+-----------+-----------+
| cbt    | chr2L     |    476437 |
| ash2   | chr3R     |  20477248 |
| HNF1A  | chr12     | 121416548 |
| MYC    | chr8      | 128748314 |
+--------+-----------+-----------+
4 rows in set (0,00 sec)

mysql> SELECT nom,cromosoma,inici FROM gens ORDER BY nom;

+--------+-----------+-----------+
|  nom   | cromosoma |   inici   |
+--------+-----------+-----------+
| ash2   | chr3R     |  20477248 |
| cbt    | chr2L     |    476437 |
| HNF1A  | chr12     | 121416548 |
| MYC    | chr8      | 128748314 |
+--------+-----------+-----------+
4 rows in set (0,00 sec)

mysql> SELECT nom,cromosoma,inici FROM gens
    -> ORDER BY nom DESC;

+--------+-----------+-----------+
|  nom   | cromosoma |   inici   |
+--------+-----------+-----------+
| MYC    | chr8      | 128748314 |
| HNF1A  | chr12     | 121416548 |
| cbt    | chr2L     |    476437 |
| ash2   | chr3R     |  20477248 |
+--------+-----------+-----------+
4 rows in set (0,01 sec)
```

Figura 28. Ordenar els registres d'una taula.
Font: elaboració pròpia.

Quan es treballa amb taules que contenen milers d'elements, resulta convenient mostrar inicialment només els primers registres per comprovar el correcte funcionament de la consulta. La funció `LIMIT` permet mostrar exclusivament els primers *n* registres de la consulta en execució:

```
mysql> SELECT * FROM gens LIMIT 1;

|  nom   | cromosoma | cadena |   inici   |   final   | proteina |    especie      |
+--------+-----------+--------+-----------+-----------+----------+-----------------+
| ash2   | chr3R     | +      | 20477248 | 20479098 | NP_733023 | D. melanogaster |
+--------+-----------+--------+-----------+-----------+----------+-----------------+
1 row in set (0,00 sec)
```

Figura 29. Mostrar un fragment de la consulta.
Font: elaboració pròpia.

1. Bases de dades relacionals

1.8. Consultes bàsiques: filtres i condicions

L'ordre SELECT permet també extreure de les taules únicament aquells registres que posseeixen certes propietats. Per especificar el filtre a realitzar sobre el contingut d'una taula, cal afegir-hi la clàusula WHERE:

```
SELECT camp1,camp2,...,campn FROM taula WHERE condicio;
```

Figura 30. Sintaxi bàsica de l'ordre SELECT amb condicions.
Font: elaboració pròpia.

La condició pot ser simple o composta, avaluant-se sobre un o més atributs de diverses taules. Els operadors de comparació més habituals es mostren a la taula 3.

Taula 3. Operadors de comparació en consultes de MySQL.

Operador	Significat
=,<>	Igual/diferent
<,>	Menor/major
<=,>=	Menor/major o igual
LIKE	Recerca d'un patró de text
NOT	Negació d'una condició
AND/OR	Condicions combinades
REGEXP	Expressió regular

Font: elaboració pròpia.

Els operadors numèrics també resulten molt útils per buscar registres en un rang concret de dates del calendari.

Provarem aquests operadors per realitzar consultes més concretes sobre la nostra base de dades **cataleg**. En primer lloc, podem interrogar la base de dades sobre els gens ubicats en el fil positiu de la cadena d'ADN en qualsevol espècie, preguntar per aquells que no pertanyen a la nostra espècie o buscar els gens anotats abans del primer milió de bases en qualsevol cromosoma:

```
mysql> SELECT * FROM gens WHERE cadena LIKE '+';

+--------+-----------+---------+-----------+-----------+-----------+-----------------+
|   nom  | cromosoma | cadena  |   inici   |   final   | proteina  |     especie     |
+--------+-----------+---------+-----------+-----------+-----------+-----------------+
| ash2   | chr3R     | +       |  20477248 |  20479098 | NP_733023 | D. melanogaster |
| HNF1A  | chr12     | +       | 121416548 | 121440312 | NP_000536 | H. sapiens      |
| MYC    | chr8      | +       | 128748314 | 128753678 | NP_002458 | H. sapiens      |
+--------+-----------+---------+-----------+-----------+-----------+-----------------+
3 rows in set (0,00 sec)

mysql> SELECT * FROM gens WHERE especie NOT LIKE 'H. sapiens';

+--------+-----------+---------+----------+----------+-----------+-----------------+
|   nom  | cromosoma | cadena  |   inici  |   final  | proteina  |     especie     |
+--------+-----------+---------+----------+----------+-----------+-----------------+
| ash2   | chr3R     | +       | 20477248 | 20479098 | NP_733023 | D. melanogaster |
| cbt    | chr2L     | -       |   476437 |   479046 | NP_722636 | D. melanogaster |
+--------+-----------+---------+----------+----------+-----------+-----------------+
2 rows in set (0,00 sec)

mysql> SELECT * FROM gens WHERE inici <= 1000000;

+--------+-----------+---------+--------+--------+-----------+-----------------+
|   nom  | cromosoma | cadena  | inici  | final  | proteina  |     especie     |
+--------+-----------+---------+--------+--------+-----------+-----------------+
| cbt    | chr2L     | -       | 476437 | 479046 | NP_722636 | D. melanogaster |
+--------+-----------+---------+--------+--------+-----------+-----------------+
1 row in set (0,00 sec)
```

Figura 31. Consultes amb una condició.
Font: elaboració pròpia.

La clàusula es pot complementar amb el modificador %, que actua de comodí en les expressions alfanumèriques. A continuació, seleccionem només registres que pertanyen al genoma de la mosca:

```
mysql> SELECT * FROM gens WHERE especie LIKE '%melano%';

+--------+-----------+---------+----------+----------+-----------+-----------------+
|   nom  | cromosoma | cadena  |   inici  |   final  | proteina  |     especie     |
+--------+-----------+---------+----------+----------+-----------+-----------------+
| ash2   | chr3R     | +       | 20477248 | 20479098 | NP_733023 | D. melanogaster |
| cbt    | chr2L     | -       |   476437 |   479046 | NP_722636 | D. melanogaster |
+--------+-----------+---------+----------+----------+-----------+-----------------+
2 rows in set (0,00 sec)
```

Figura 32. Consultes sobre patrons de text.
Font: elaboració pròpia.

També podem combinar preguntes sobre valors de diferents tipus. Per exemple, si desitgem esbrinar quants gens de la mosca de la fruita estan anotats al fil positiu de la cadena d'ADN:

```
mysql> SELECT * FROM gens WHERE especie LIKE '%melano%'
    -> AND cadena LIKE '+';

+--------+-----------+---------+----------+----------+-----------+-----------------+
|   nom  | cromosoma | cadena  |   inici  |   final  | proteina  |     especie     |
+--------+-----------+---------+----------+----------+-----------+-----------------+
| ash2   | chr3R     | +       | 20477248 | 20479098 | NP_733023 | D. melanogaster |
+--------+-----------+---------+----------+----------+-----------+-----------------+
1 row in set (0,00 sec)
```

Figura 33. Consultes amb dues condicions.
Font: elaboració pròpia.

1. Bases de dades relacionals

1.9. Consultes avançades: agrupacions

Mitjançant el desglossament de dades d'una taula, en funció d'algun camp concret, podem calcular estadístiques sobre cada categoria. L'ordre GROUP BY permet realitzar agrupacions de les dades de les taules segons els criteris que establim, i és possible combinar aquestes classificacions amb operadors d'agregació com COUNT, MAX, MIN, AVG o SUM.

```
SELECT camp1,camp2,...,campn FROM taula GROUP BY atribut;
```

Figura 34. Sintaxi de l'ordre SELECT sobre grups.
Font: elaboració pròpia.

Per exemple, demanem les espècies presents a la nostra base de dades:

```
mysql> SELECT especie FROM gens GROUP BY especie;

+----------------+
| especie        |
+----------------+
| D.melanogaster |
| H.sapiens      |
+----------------+
2 rows in set (0,00 sec)
```

Figura 35. Dades agrupades per espècie.
Font: elaboració pròpia.

Posteriorment, podem comptar el nombre exacte d'exemples de cada espècie:

```
mysql> SELECT especie,COUNT(*) FROM gens GROUP BY especie;

+----------------+----------+
| especie        | COUNT(*) |
+----------------+----------+
| D. melanogaster |        2 |
| H. sapiens      |        2 |
+----------------+----------+
2 rows in set (0,00 sec)
```

Figura 36. Nombre d'espècies emmagatzemades a la taula *gens*.
Font: elaboració pròpia.

Ara obtenim les estadístiques bàsiques sobre la longitud dels gens:

```
mysql> SELECT especie,cromosoma,inici,final,final-inici
    -> FROM gens;

+----------------+-----------+----------+----------+--------------+
| especie        | cromosoma | inici    | final    | final-inici  |
+----------------+-----------+----------+----------+--------------+
| D. melanogaster | chr3R     |  20477248 |  20479098 |         1850 |
| D. melanogaster | chr2L     |    476437 |    479046 |         2609 |
| H. sapiens      | chr12     | 121416548 | 121440312 |        23764 |
| H. sapiens      | chr8      | 128748314 | 128753678 |         5364 |
+----------------+-----------+----------+----------+--------------+
4 rows in set (0,00 sec)

mysql> SELECT especie,AVG(final-inici),MIN(final-inici),
    -> MAX(final-inici) FROM gens GROUP BY especie;

+----------------+------------------+------------------+------------------+
| especie        | AVG(final-inici) | MIN(final-inici) | MAX(final-inici) |
+----------------+------------------+------------------+------------------+
| D. melanogaster |        2229.5000 |             1850 |             2609 |
| H. sapiens      |       14564.0000 |             5364 |            23764 |
+----------------+------------------+------------------+------------------+
2 rows in set (0,00 sec)
```

Figura 37. Càlcul de mitjanes a la base de dades *cataleg*.
Font: elaboració pròpia.

1. Bases de dades relacionals

1.10. Consultes avançades: consultes multitaula

Per aprofitar al màxim el model relacional i generar nou coneixement de les dades existents resulta essencial combinar informació de diverses taules. Mitjançant un atribut que dues taules posseeixin en comú, aquesta operació resulta extremadament senzilla amb SQL. Mentre efectuem una consulta amb l'ordre SELECT, hem d'emprar la clàusula JOIN especificant l'atribut compartit per ambdues taules. Quan es referencien atributs de dues o més taules en la mateixa consulta, cal utilitzar la sintaxi `nombre_taula.nom_ atribut` per especificar clarament l'origen de cada atribut. És possible afegir-hi condicions sobre altres atributs de les taules amb la clàusula WHERE.

Donades dues taules que denominarem **taula1** i **taula2**, que posseeixen un atribut comparable (**taula1.x** i **taula2.y**), la sintaxi de l'ordre **JOIN** per obtenir tant els valors en comú com els valors presents exclusivament en la primera o en la segona taula, respectivament, és la següent:

```
SELECT taula1.x,taula2.y
FROM taula1 JOIN taula2
ON taula1.x=taula2.y
WHERE condicions;
-----------------------------------------------
SELECT taula1.x,taula2.y
FROM taula1 LEFT JOIN taula2
ON taula1.x=taula2.y
WHERE condicions;
-----------------------------------------------
SELECT taula1.x,taula2.y
FROM taula1 RIGHT JOIN taula2
ON taula1.x=taula2.y
WHERE condicions;
```

Figura 38. Sintaxi de la clàusula JOIN.
Font: elaboració pròpia.

Per treballar amb més de dues taules, podem generalitzar la mateixa sintaxi (per exemple, taula1 JOIN (taula2,3,4)).

Abans de procedir a executar consultes sobre les taules de la base de dades, proposem posar a prova el funcionament de la clàusula JOIN sobre un exemple més simple.

Generarem dues taules que denominarem **taula1** i **taula2**, amb un únic atribut. Posteriorment, poblarem ambdues taules amb una sèrie de valors tals que resultarà molt senzill mostrar el conjunt complet de combinacions a l'hora de comparar les dues taules.

Primer procedim a crear les dues taules:

```
mysql> CREATE TABLE taula1
    -> (x VARCHAR(10));

Query OK, 0 rows affected (0,5 sec)

mysql> CREATE TABLE taula2
    -> (y VARCHAR(10));

Query OK, 0 rows affected (0,5 sec)
```

Figura 39. Crear dues taules per combinar amb la clàusula JOIN.
Font: elaboració pròpia.

> Us animem a reproduir al vostre ordinador les ordres de SQL presentades en aquest apartat.

Ara crearem dos fitxers de text senzills amb tres valors cadascun: (1,2,3) i (3,4,5). D'aquesta manera, podrem estudiar detalladament la classe de consulta de MySQL que necessitem per recuperar els valors comuns entre ambdues taules (3) o, alternativament, els valors que únicament apareixen a la primera (1 i 2) o a la segona taula (4 i 5).

```
1
2
3

-----------------------------------------------

3
4
5
```

Figura 40. Les dades .txt i dades2.txt.
Font: elaboració pròpia.

Finalment, poblem les dues taules utilitzant ambdós fitxers:

```
mysql> LOAD DATA LOCAL INFILE 'dades1.txt' INTO TABLE taula1;

Query OK, 3 rows affected (0,67 sec)
Records: 3  Deleted: 0  Skipped: 0  Warnings: 0

mysql> LOAD DATA LOCAL INFILE 'dades2.txt' INTO TABLE taula2;

Query OK, 3 rows affected (0,43 sec)
Records: 3  Deleted: 0  Skipped: 0  Warnings: 0
```

Figura 41. Poblem les dues taules per combinar amb la clàusula JOIN.
Font: elaboració pròpia.

Ja estem en disposició d'aprofundir en el funcionament de les consultes que inclouen la clàusula JOIN. Si no hi afegim cap opció, aquesta ordre genera totes les parelles possibles el primer element de les quals pertany a la primera taula i el segon element pertany a la segona:

```
mysql> SELECT taula1.x,taula2.y
    -> FROM taula1 JOIN taula2;

+------+------+
| x    | y    |
+------+------+
| 1    | 3    |
| 2    | 3    |
| 3    | 3    |
| 1    | 4    |
| 2    | 4    |
| 3    | 4    |
| 1    | 5    |
| 2    | 5    |
| 3    | 5    |
+------+------+
9 rows in set (0,00 sec)
```

Figura 42. Combinació amb JOIN de dues taules per obtenir totes les combinacions.
Font: elaboració pròpia.

El nostre primer objectiu en una comparació és localitzar els elements comuns. Per tal de fer-ho n'hi ha prou amb incorporar la clàusula **ON** a la consulta i especificar l'atribut compartit per les dues taules. Això filtrarà aquelles parelles del resultat anterior que no compleixin aquesta propietat, i es demostrarà allò que busquem:

```
mysql> SELECT taula1.x,taula2.y
    -> FROM taula1 JOIN taula2
    -> ON taula1.x=taula2.y;

+------+------+
| x    | y    |
+------+------+
| 3    | 3    |
+------+------+
1 row in set (0,00 sec)
```

Figura 43. Combinació amb JOIN de dues taules amb la clàusula ON.
Font: elaboració pròpia.

En determinats casos, en lloc de la llista dels valors comuns, estarem interessats també en aquells valors d'una o altra taula que no pertanyen a l'altra. Per això hem de modificar el comportament de l'ordre JOIN amb les clàusules LEFT o RIGHT. Si realitzem una unió per la part esquerra, recuperarem un llistat dels registres de la primera taula juntament amb el seu registre equivalent a la segona. En el cas que aquest valor anàleg no existís, MySQL ho indicarà amb el valor NULL. Si, per contra, realitzem la unió per la dreta, obtindrem un llistat dels valors de la segona taula sota les mateixes condicions.

```
mysql> SELECT taula1.x,taula2.y
    -> FROM taula1 LEFT JOIN taula2
    -> ON taula1.x=taula2.y;

+------+------+
| x    | y    |
+------+------+
| 3    | 3    |
| 1    | NULL |
| 2    | NULL |
+------+------+
3 rows in set (0,00 sec)

mysql> SELECT taula1.x,taula2.y
    -> FROM taula1 RIGHT JOIN taula2
    -> ON taula1.x=taula2.y;

+------+------+
| x    | y    |
+------+------+
| 3    | 3    |
| NULL | 4    |
| NULL | 5    |
+------+------+
3 rows in set (0,00 sec)
```

Figura 44. Combinació amb JOIN de dues taules amb les clàusules LEFT i RIGHT.
Font: elaboració pròpia.

Per acabar de refinar el resultat, hem de mantenir exclusivament els valors que només estan en una taula. Per aconseguir-ho, podem afegir al final de la consulta una condició WHERE que exigeixi que el valor no estigui present a l'altra taula. La clàusula IS realitza l'avaluació de l'expressió que es troba a continuació, per acabar responent amb un valor booleà (cert o fals).

```
mysql> SELECT taula1.x,taula2.y
    -> FROM taula1 LEFT JOIN taula2
    -> ON taula1.x=taula2.y
    -> WHERE taula2.y IS NULL;

+------+------+
| x    | y    |
+------+------+
| 1    | NULL |
| 2    | NULL |
+------+------+
2 rows in set (0,00 sec)

mysql> SELECT taula1.x,taula2.y
    -> FROM taula1 RIGHT JOIN taula2
    -> ON taula1.x=taula2.y
    -> WHERE taula1.x IS NULL;

+------+------+
| x    | y    |
+------+------+
| NULL | 4    |
| NULL | 5    |
+------+------+
2 rows in set (0,00 sec)
```

Figura 45. Combinació amb JOIN de dues taules emprant una condició.
Font: elaboració pròpia.

Ara ja estem en condicions de realitzar consultes sobre dues o més taules del nostre catàleg de gens. Per exemple, podem preguntar per aquells gens humans per als quals s'ha documentat una anotació funcional de caràcter experimental:

```
mysql> SELECT gens.nom,gens.especie,
    -> anotacions.funcio,anotacions.origen
    -> FROM gens JOIN anotacions
    -> ON gens.nom=anotacions.nom
    -> WHERE anotacions.origen='experimental'
    -> AND gens.especie='H. sapiens';

+--------+-----------+-----------+-------------+
| nom    |  especie  | funcio    |    origen   |
+--------+-----------+-----------+-------------+
| HNF1A  | H. sapiens | GO:0003700 | Experimental |
| MYC    | H. sapiens | GO:0003700 | Experimental |
+--------+-----------+-----------+-------------+
2 rows in set (0,00 sec)
```

Figura 46. Consultes sobre el catàleg de gens utilitzant l'ordre JOIN.
Font: elaboració pròpia.

1. Bases de dades relacionals

1.11. Consultes avançades: subconsultes

En ocasions desitgem realitzar un tipus de pregunta sobre les nostres dades, però no és factible perquè l'organització en taules escollida no ho permet. Per resoldre aquest problema, MySQL permet enniuar una consulta dins d'una altra, amb l'objectiu d'utilitzar la pregunta interior per donar-li la forma apropiada a les dades, que podran ser tractades posteriorment mitjançant la consulta exterior.

Sintaxi bàsica d'una subconsulta:

```
SELECT llista_columnes
FROM nombre_taula
WHERE condició = (SELECT llista_columnes2
FROM nombre_taula2
WHERE condicions);
```

Sintàcticament, des del punt de vista de la consulta principal, la subconsulta interior exercirà el paper d'una taula convencional. Per aquesta raó, és possible assignar un nom tant a la consulta interior com als atributs dels resultats que se'n desprendran. Per a això emprarem la clàusula **AS**, que permet associar un nom a un grup d'operacions o atributs en SQL. El nom emprat per a aquests atributs resulta útil per referir-s'hi des de la consulta exterior.

```
SELECT subconsulta.valor_1, ..., subconsulta.valor_n FROM
   (SELECT atribut_1 AS valor_1, ..., atribut_n AS valor_n
    FROM taula GROUP BY atribut_i) AS subconsulta;
```

Figura 47. Sintaxi de les subconsultes.
Font: elaboració pròpia.

Per exemplificar la classe d'escenari on les subconsultes resulten potencialment interessants, imaginem una taula genèrica anomenada *taula* amb dos atributs, que denominarem *clase* i *subclsse*. Cada registre d'aquesta taula pertany a una classe general, i dins d'aquesta classe, a una subclasse més específica. Suposem que ens agradaria calcular la mitjana de subclasses diferents, classe per classe, que han estat utilitzades per etiquetar cada registre. Per obtenir la resposta, definirem una subconsulta que rebrà el nom de *contador*. Aquesta subpregunta agruparà les dades per classes per comptar el nombre total de subclasses assignat als registres de cada classe principal. Finalment, la consulta exterior simplement haurà de calcular la mitjana dels totals calculats per la subconsulta.

```
classe_1 subclasse_x
classe_1 subclasse_y
classe_1 subclasse_z
classe_2 subclasse_a
classe_2 subclasse_b
classe_3 subclasse_n
...
-----------------------------------------------

SELECT AVG(comptador.totals) FROM

(SELECT count(subclasse) AS totals

 FROM taula GROUP BY classe) AS comptador;
```

Figura 48. Emprar una subconsulta dins d'una consulta principal.
Font: elaboració pròpia.

1. Bases de dades relacionals

1.12. Altres utilitats de MySQL

Juntament amb l'inventari d'ordres que interactuen amb la nostra base de dades emprant el llenguatge SQL, el SGBD MySQL proporciona un conjunt d'aplicacions elementals per assistir-nos a l'hora de treballar amb el sistema.

Per exemple, l'ordre `help` mostra per pantalla un breu manual d'ajuda sobre cada instrucció de MySQL.

Per poder paginar, pantalla a pantalla, sobre les entrades del manual hem d'executar abans l'aplicació **pager**. Aquesta ordre ens permet vincular una aplicació de paginació del terminal de Linux amb l'intèrpret de MySQL (per exemple, el programa *more*).

```
mysql> pager more;

PAGER set to 'more'

mysql>  help DROP TABLE;

Name: 'DROP TABLE'
Description:
Syntax:
DROP [TEMPORARY] TABLE [IF EXISTS]
    tbl_name [, tbl_name] ...
    [RESTRICT | CASCADE]

DROP TABLE removes one or more tables. You must have the DROP privilege
for each table. All table data and the table definition are removed, so
be careful with this statement! If any of the tables named in the
argument list do not exist, MySQL returns an error indicating by name
which nonexisting tables it was unable to drop, but it also drops all
of the tables in the list that do exist.

--More--
```

Figura 49. Mostra del manual d'ajut de MySQL.
Font: elaboració pròpia.

L'ordre `source` permet executar fitxers de text que inclouen ordres MySQL amb la seqüència d'instruccions precises per realitzar una determinada tasca. Per exemple, podem editar un fitxer de text des del nostre terminal amb les primeres ordres que executem en entrar en el sistema:

```
USE cataleg;
SHOW TABLES;
DESCRIBE gens;
----------------------------------------
mysql> source ordres.sql;

Database changed

+--------------------+
| Tables_in_cataleg |
+--------------------+
| anotacions         |
| funcions           |
| gens               |
| genomes             |
+--------------------+
4 rows in set (0,00 sec)

+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| nom       | varchar(20)  | NO   | PRI | NULL    |       |
| cromosoma | varchar(5)   | YES  |     | NULL    |       |
| cadena    | varchar(1)   | YES  |     | NULL    |       |
| inici     | int(11)      | YES  |     | NULL    |       |
| final     | int(11)      | YES  |     | NULL    |       |
| proteina  | varchar(20)  | YES  |     | NULL    |       |
| especie   | varchar(100) | YES  | MUL | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
7 rows in set (0,00 sec)
```

Figura 50. Execució del fitxer d'ordres comandos.sql de MySQL.
Font: elaboració pròpia.

L'ordre `status` permet veure la configuració del sistema:

```
mysql> status;

mysql  Ver 14.14 Distrib 5.7.17, for Linux (i686) using  EditLine wrapper
Connection id:          7
Current database:       catalogo
Current user:           eblanco@localhost
SSL:                    Not in use
Current pager:          more
Using outfile:          ''
Using delimiter:        ;
Server version:         5.7.17-0ubuntu0.16.04.1 (Ubuntu)
Protocol version:       10
Connection:             Localhost via UNIX socket
Server characterset:    latin1
Db      characterset:   latin1
Client characterset:    utf8
Conn.   characterset:   utf8
UNIX socket:            /var/run/mysqld/mysqld.sock
Uptime:                 4 hours 14 min 12 sec
Threads: 1  Questions: 101  Slow queries: 0  Opens: 129
Flush tables: 1  Open tables: 42  Queries per second avg: 0.006
```

Figura 51. Mostra de la configuració actual de MySQL.
Font: elaboració pròpia.

Si en algun instant necessitem accedir al terminal de Linux, podem emprar l'ordre **system** per invocar les seves ordres des de l'interior de MySQL:

```
mysql> system (ls /home/eblanco);

Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos

mysql> system (cat ordres.sql);

USE cataleg;

SHOW TABLES;

DESCRIBE gens;
```

Figura 52. Execució de l'intèrpret d'ordres de Linux a MySQL.
Font: elaboració pròpia.

1. Bases de dades relacionals

1.13. Còpies de seguretat i restauració de BBDD

És altament recomanable dur a terme còpies de seguretat de les nostres dades amb certa periodicitat. En el cas de les bases de dades gestionades amb MySQL, el programa **mysqldump**, executat des del terminal de Linux, realitza un bolcat complet del seu contingut cap a un fitxer de text. Posteriorment, en cas de ser necessari, aquest fitxer d'ordres pot ser executat amb la instrucció **source** per regenerar la base de dades completa, creant automàticament les taules i inserint-hi els registres existents en aquell moment:

```
% mysqldump -vp cataleg > cataleg.sql

Enter password:
-- Connecting to localhost...
-- Retrieving table structure for table anotaciones...
-- Sending SELECT query...
-- Retrieving rows...
-- Retrieving table structure for table funciones...
-- Sending SELECT query...
-- Retrieving rows...
-- Retrieving table structure for table genes...
-- Sending SELECT query...
-- Retrieving rows...
-- Retrieving table structure for table genomas...
-- Sending SELECT query...
-- Retrieving rows...
-- Disconnecting from localhost...

% more cataleg.sql

-- MySQL dump 10.13 Distrib 5.7.17, for Linux (i686)
-- Host: localhost Database: cataleg
-- Server version 5.7.17-0ubuntu0.16.04.1
...
CREATE TABLE `anotacions` (
...
INSERT INTO `anotacions` VALUES ('ash2','GO:0003700','Computacional'),...
...
```

Figura 53. Realitzar una còpia de seguretat amb el programa mysqldump.
Font: elaboració pròpia.

També podem restaurar una base de dades MySQL des del terminal a partir d'un fitxer *backup* així:

```
mysql -u usuari -p basededades < basededades.sql
```

1. Bases de dades relacionals

1.14. Exemple pràctic: gestió del catàleg de gens humans

Per demostrar com extreure coneixement útil d'una base de dades relacional, us proposem analitzar amb MySQL el contingut d'un catàleg de gens humans. Un gen és un fragment d'ADN ubicat en el genoma que conté la informació precisa per sintetitzar una molècula d'ARN. En els organismes eucariotes, un gen està constituït per una successió de fragments útils denominats *exons*. En una proporció significativa dels gens humans hi ha diverses combinacions alternatives d'exons, donant lloc a diferents formes alternatives d'un mateix gen, denominades *transcrits alternatius*. Per codificar la informació relativa a la localització dels gens en el genoma és freqüent utilitzar fitxers de text tabulat. Cada línia d'aquests fitxers conté els valors dels atributs que caracteritzen un transcrit d'un determinat gen. Bàsicament, un transcrit d'un gen posseeix una localització concreta, identificada per un cromosoma, una posició inicial/final i una direcció de lectura. Altres característiques que podem recuperar sobre un transcrit són el seu codi, el nom del gen, el nombre d'exons o les seves coordenades exactes.

> **Vegeu també**
>
> Per revisar els conceptes de *genoma*, *cromosoma*, *gen* i *proteïna* us recomanem l'assignatura Fonaments de biologia molecular.

El navegador genòmic d'UCSC representa gràficament els diferents tipus d'anotacions existents sobre el genoma humà en forma de centenars de pistes. Per administrar eficientment aquest elevat volum d'informació, una còpia del SGBD MySQL està funcionant de forma transparent als milers d'usuaris que cada dia visiten aquest servidor. D'aquesta manera, en el cas que volguem reproduir una pista al nostre ordinador, disposem a la secció de descàrregues d'un fitxer SQL per ser executat amb l'ordre `source` i un fitxer de text amb el conjunt de dades que s'han d'importar amb la instrucció `LOAD DATA`. En aquest exercici utilitzarem l'anotació dels gens humans distribuïda pel consorci **RefSeq** per al genoma humà. Aquest format és comú a totes les espècies subministrades pel navegador.

> **Vegeu també**
>
> És possible aprofundir sobre el funcionament dels navegadors genòmics en l'assignatura Genòmica computacional.

Ara procedirem a descarregar-nos els dos fitxers associats a la pista **refGene**, que conté el catàleg de gens humans anotats pel consorci **RefSeq**, en la seva versió **hg38**.

Per a això, hem d'utilitzar l'ordre `wget` per transferir tots dos fitxers al nostre terminal.

`wget` http://hgdownload.soe.ucsc.edu/goldenPath/hg38/database/refGene.sql

`wget` http://hgdownload.soe.ucsc.edu/goldenPath/hg38/database/refGene.txt.gz

Mostrem a continuació el contingut del fitxer **refGene.sql** que realitza la creació de la taula **refGene**. Els atributs que consultarem amb més freqüència seran (figura 54): **name** (codi del transcrit), **chrom** (cromosoma), **strand** (cadena), **txStart** i **txEnd** (coordenades d'inici i final), **exonCount** (nombre d'exons) i **name2** (nom del gen).

És important no confondre els camps de **name** i **name2**: un gen pot tenir diversos transcrits, però un transcrit únicament pertany a un gen.

```
CREATE TABLE 'refGene' (
  'bin' smallint(5) unsigned NOT NULL,
  'name' varchar(255) NOT NULL,
  'chrom' varchar(255) NOT NULL,
  'strand' char(1) NOT NULL,
  'txStart' int(10) unsigned NOT NULL,
  'txEnd' int(10) unsigned NOT NULL,
  'cdsStart' int(10) unsigned NOT NULL,
  'cdsEnd' int(10) unsigned NOT NULL,
  'exonCount' int(10) unsigned NOT NULL,
  'exonStarts' longblob NOT NULL,
  'exonEnds' longblob NOT NULL,
  'score' int(11) DEFAULT NULL,
  'name2' varchar(255) NOT NULL,
  'cdsStartStat' enum('none','unk','incmpl','cmpl') NOT NULL,
  'cdsEndStat' enum('none','unk','incmpl','cmpl') NOT NULL,
  'exonFrames' longblob NOT NULL,
  KEY 'chrom' ('chrom','bin'),
  KEY 'name' ('name'),
  KEY 'name2' ('name2')
) ENGINE-MyISAM DEFAULT CHARSET-latin1;
```

Figura 54. Atributs dels transcrits anotats pel consorci RefSeq.
Font: elaboració pròpia.

Passarem ara a visualitzar amb el terminal el segon fitxer **refGene.txt**. Aquest arxiu conté les dades del catàleg complet de gens anotats en el genoma humà. Hem de carregar aquesta informació a la nostra base de dades un cop estigui creada la taula **refGene**. En el context d'aquest exercici, cada registre conté informació sobre el transcrit d'un determinat gen. En el cas que un gen posseeixi diversos transcrits, cadascun es codifica en registres separats (cadascú amb el seu propi codi i les seves corresponents coordenades).

En primer lloc, hem de descomprimir el fitxer amb l'ordre **gzip**.

```
% gzip -d refGene.txt.gz

% head -5 refGene.txt
585 NR_046018 chr1 + 11873 14409 14409 14409 3 11873,12612,13220, 12227,12721,14409, 0 DDX11L1
unk unk -1,-1,-1,

585 NR_024540 chr1 - 14361 29370 29370 29370 11 14361,14969,15795,16606,16857,17232,17605,17914,
18267,24737,29320, 14829,15038,15947,16765,17055,17368,17742,18061,18366,24891,29370, 0 WASH7P
unk unk -1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,

932 NR_104645 chrX + 45505387 45523644 45523644 45523644 3 45505387,45510496,45521607, 45505465,
45510595,45523644, 0 LINC01204 unk unk -1,-1,-1,

1078 NR_104148 chr7 + 64666082 64687830 64687830 64687830 4 64666082,64669036,64679176,64684334,
64666285,64669178,64679336,64687830, 0 ZNF107 unk unk -1,-1,-1,-1,

103 NR_120408 chr14 + 31561384 31861223 31861223 31861223 10 31561384,31562067,31565013,31599288,
31673354,31673483,31826628,31846470,31850118,31859117, 31561547,31562215,31565048,31599379,
31673394,31673574,31826714,31846591,31850201,31861223, 0NUBPL unk unk -1,-1,-1,-1,-1,-1,-1,-1,-1,
-1,
```

Figura 55. El catàleg refGene.txt de gens humans.
Font: elaboració pròpia.

> Les anotacions d'un genoma solen actualitzar-se freqüentment. Per aquest motiu, les dades mostrades en aquest tutorial poden variar lleugerament amb el pas del temps.

Un cop dins de l'intèrpret de MySQL, indicarem que treballarem dins de la nostra base **de dades**.

Executarem, posteriorment, el fitxer **refGene.sql** amb l'ordre `source` per crear la taula **refGene**.

Per verificar que la instrucció anterior ha funcionat correctament, podem veure el llistat d'atributs de la taula **refGene** amb l'ordre `DESCRIBE`:

```
mysql> USE cataleg;

Database changed

mysql> source refGene.sql;

Query OK, 0 rows affected (0,00 sec)

mysql> DESCRIBE refGene;

+--------------+-----------------------------------+------+-----+---------+-------+
| Field        | Type                              | Null | Key | Default | Extra |
+--------------+-----------------------------------+------+-----+---------+-------+
| bin          | smallint(5) unsigned              | NO   |     | NULL    |       |
| name         | varchar(255)                      | NO   | MUL | NULL    |       |
| chrom        | varchar(255)                      | NO   | MUL | NULL    |       |
| strand       | char(1)                           | NO   |     | NULL    |       |
| txStart      | int(10) unsigned                  | NO   |     | NULL    |       |
| txEnd        | int(10) unsigned                  | NO   |     | NULL    |       |
| cdsStart     | int(10) unsigned                  | NO   |     | NULL    |       |
| cdsEnd       | int(10) unsigned                  | NO   |     | NULL    |       |
| exonCount    | int(10) unsigned                  | NO   |     | NULL    |       |
| exonStarts   | longblob                          | NO   |     | NULL    |       |
| exonEnds     | longblob                          | NO   |     | NULL    |       |
| score        | int(11)                           | YES  |     | NULL    |       |
| name2        | varchar(255)                      | NO   | MUL | NULL    |       |
| cdsStartStat | enum('none','unk','incmpl','cmpl') | NO   |     | NULL    |       |
| cdsEndStat   | enum('none','unk','incmpl','cmpl') | NO   |     | NULL    |       |
| exonFrames   | longblob                          | NO   |     | NULL    |       |
+--------------+-----------------------------------+------+-----+---------+-------+
16 rows in set (0,14 sec)
```

Figura 56. Creació de la taula refGene.
Font: elaboració pròpia.

> Assumirem que ambdós fitxers (refGene.sql i refGene.txt) estan guardats en la mateixa carpeta de treball des de la qual hem invocat el programa MySQL, anteriorment.

El segon pas consisteix a poblar la taula amb les anotacions dels gens humans que hem descarregat dins del fitxer **refGene.txt**. Fent servir l'ordre **LOAD DATA** podem bolcar tot el contingut a la taula **refGene**:

```
mysql> LOAD DATA LOCAL INFILE 'refGene.txt' INTO TABLE refGene;

Query OK, 69853 rows affected (2,54 sec)
Records: 69853  Deleted: 0  Skipped: 0  Warnings: 0
```

Figura 57. Poblar la taula refGene.
Font: elaboració pròpia.

Ens trobem en condicions de començar a interrogar la base de dades. Recordem, novament, que cada registre de la taula **refGene** alberga la informació associada a l'espai d'un gen en particular. Igualment, és important tenir en compte que una elevada fracció dels gens humans posseeix dos o més transcrits alternatius. La nostra missió, a continuació, és mostrar l'enorme potencial de les consultes de SQL a l'hora d'extreure nou coneixement biològic de les dades emmagatzemades a les taules de la nostra base de dades.

Començarem mostrant els primers registres de la nostra taula, incloent-hi únicament diversos dels seus atributs per afavorir la llegibilitat dels valors dels registres per pantalla:

```
mysql> SELECT name2,name,chrom,strand,txStart,txEnd,exonCount
    -> FROM refGene ORDER BY name2 LIMIT 10;

+---------+-------------+-------+--------+----------+----------+-----------+
| name2   | name        | chrom | strand | txStart  | txEnd    | exonCount |
+---------+-------------+-------+--------+----------+----------+-----------+
| A1BG    | NM_130786   | chr19 | -      | 58346805 | 58353499 |         8 |
| A1BG-AS1| NR_015380   | chr19 | +      | 58351969 | 58355183 |         4 |
| A1CF    | NM_001198819| chr10 | -      | 50799408 | 50885675 |        15 |
| A1CF    | NM_014576   | chr10 | -      | 50799408 | 50885675 |        13 |
| A1CF    | NM_138932   | chr10 | -      | 50799408 | 50885675 |        13 |
| A1CF    | NM_001198820| chr10 | -      | 50799408 | 50885675 |        14 |
| A1CF    | NM_001198818| chr10 | -      | 50799408 | 50885675 |        14 |
| A1CF    | NM_138933   | chr10 | -      | 50799408 | 50885675 |        13 |
| A2M     | NM_001347423| chr12 | -      |  9067707 |  9116229 |        37 |
| A2M     | NM_000014   | chr12 | -      |  9067707 |  9116229 |        36 |
+---------+-------------+-------+--------+----------+----------+-----------+
10 rows in set (0,00 sec)
```

Figura 58. Mostra del contingut de la taula refGene.
Font: elaboració pròpia.

Atès que cada registre conté la informació d'un transcrit, el nombre de transcrits coneguts en el genoma humà coincidirà amb el nombre de registres emmagatzemats a la taula **refGene**. Aquest comptatge és senzill:

```
mysql> SELECT COUNT(*) FROM refGene;

+----------+
| COUNT(*) |
+----------+
|    69853 |
+----------+
1 row in set (0,00 sec)
```

Figura 59. Comptar els transcrits de la taula refGene.
Font: elaboració pròpia.

També podem comptar fàcilment el nombre total de gens codificats en el genoma humà. Si un gen posseeix diversos transcrits alternatius, llavors trobarem diversos registres en el nostre catàleg que posseeixen un valor diferent de l'atribut **name**, però que comparteixen el mateix valor per a l'atribut **name2**. Per tant, emprant la clàusula DISTINCT sobre aquest últim atribut, comptarem una única vegada cada gen de la nostra taula, encara que posseeixi diverses formes alternatives:

```
mysql> SELECT COUNT(DISTINCT name2) FROM refGene;

+----------------------+
| COUNT(DISTINCT name2) |
+----------------------+
|                27656 |
+----------------------+
1 row in set (0,07 sec)
```

Figura 60. Comptar els gens de la taula refGene.
Font: elaboració pròpia.

Si agrupem els registres de la taula per l'atribut **name2**, podem elaborar un inventari del nombre de transcrits alternatius anotats per a cada gen.

```
mysql> SELECT name2,COUNT(name2)
    -> FROM refGene GROUP BY name2 LIMIT 10;

+----------+--------------+
| name2    | COUNT(name2) |
+----------+--------------+
| A1BG     |            1 |
| A1BG-AS1 |            1 |
| A1CF     |            6 |
| A2M      |            4 |
| A2M-AS1  |            3 |
| A2ML1    |            2 |
| A2MP1    |            1 |
| A3GALT2  |            1 |
| A4GALT   |            3 |
| A4GNT    |            1 |
+----------+--------------+
10 rows in set (0,00 sec)
```

Figura 61. Comptar el nombre de transcrits de cada gen de la taula refGene.
Font: elaboració pròpia.

Podem obtenir resultats interessants aplicant la clàusula WHERE sobre els atributs de cada registre. Per exemple, imaginem que desitgem conèixer el nombre de transcrits ubicats en cada cadena de la molècula d'ADN:

```
mysql> SELECT COUNT(*) FROM refGene WHERE strand LIKE '+';

+----------+
| COUNT(*) |
+----------+
|    35724 |
+----------+
1 row in set (0,10 sec)

mysql> SELECT COUNT(*) FROM refGene WHERE strand LIKE '-';

+----------+
| COUNT(*) |
+----------+
|    34129 |
+----------+
1 row in set (0,09 sec)
```

Figura 62. Comptar en una cadena d'ADN.
Font: elaboració pròpia.

També podem comptar el nombre de transcrits localitzats en un cromosoma:

```
mysql> SELECT COUNT(*) FROM refGene WHERE chrom LIKE 'chr21';

+----------+
| COUNT(*) |
+----------+
|      961 |
+----------+
1 row in set (0,00 sec)
```

Figura 63. Comptar els transcrits d'un cromosoma.
Font: elaboració pròpia.

Novament, jugant amb l'atribut **name2** podem comptar el nombre de gens codificats en el mateix cromosoma:

```
mysql> SELECT COUNT(DISTINCT name2)
    -> FROM refGene WHERE chrom LIKE 'chr21';

+----------------------+
| COUNT(DISTINCT name2) |
+----------------------+
|                  408 |
+----------------------+
1 row in set (0,01 sec)
```

Figura 64. Comptar els gens d'un cromosoma.
Font: elaboració pròpia.

O identificar quins són els transcrits que posseeixen un major nombre d'exons:

```
mysql> SELECT name2,name,exonCount
    -> FROM refGene ORDER BY exonCount DESC LIMIT 10;

+-------+-------------+-----------+
| name2 | name        | exonCount |
+-------+-------------+-----------+
| TTN   | NM_001267550 |       363 |
| TTN   | NM_001256850 |       313 |
| TTN   | NM_133378    |       312 |
| TTN   | NM_133437    |       192 |
| TTN   | NM_133432    |       192 |
| TTN   | NM_003319    |       191 |
| NEB   | NM_001271208 |       183 |
| NEB   | NM_001164507 |       182 |
| NEB   | NM_001164508 |       182 |
| MUC19 | NM_173600    |       174 |
+-------+-------------+-----------+
10 rows in set (0,11 sec)
```

Figura 65. Recuperar els transcrits amb major nombre d'exons.
Font: elaboració pròpia.

També podem seleccionar aquells que posseeixen un únic exó:

```
mysql> SELECT name2,name,exonCount
    -> FROM refGene WHERE exonCount=1
    -> ORDER BY name2 LIMIT 10;

+-------------+-------------+-----------+
| name2       | name        | exonCount |
+-------------+-------------+-----------+
| AADACL2-AS1 | NR_110203   |         1 |
| ABALON      | NR_131907   |         1 |
| ABHD16B     | NM_080622   |         1 |
| ACKR1       | NM_001122951 |         1 |
| ACKR4       | NM_178445   |         1 |
| ACTBL2      | NM_001017992 |         1 |
| ACTG1P20    | NR_033926   |         1 |
| ACTG1P4     | NR_024438   |         1 |
| ACTL10      | NM_001024675 |         1 |
| ACTL7A      | NM_006687   |         1 |
+-------------+-------------+-----------+
10 rows in set (0,00 sec)
```

Figura 66. Recuperar els transcrits amb un únic exó.
Font: elaboració pròpia.

És possible calcular el nombre d'exons, de mitjana, per cada transcrit:

```
mysql> SELECT AVG(exonCount) FROM refGene;

+----------------+
| AVG(exonCount) |
+----------------+
|         9.4126 |
+----------------+
1 row in set (0,11 sec)
```

Figura 67. Calcular el nombre d'exons de mitjana per cada transcrit.
Font: elaboració pròpia.

I la longitud mitjana dels gens humans:

```
mysql> SELECT AVG(txEnd-txStart+1) FROM refGene;

+----------------------+
| AVG(txEnd-txStart+1) |
+----------------------+
|           56983.2770 |
+----------------------+
1 row in set (0,10 sec)
```

Figura 68. Calcular la longitud mitjana dels gens.
Font: elaboració pròpia.

Finalment, integrarem en aquesta anàlisi el genoma de ratolí domèstic. Descarreguem els fitxers **refGene.sql** i **refGene.txt** d'aquesta espècie en la seva versió mm9.

Per evitar sobreescriure les anotacions humanes, hem de gravar ambdós fitxers amb un nom diferent (per exemple, **refGene_mouse.sql** i **refGene_mouse.txt**). Posteriorment, cal editar el contingut del fitxer SQL per modificar el nom de la taula, per la mateixa raó (figura 69).

Després d'aquestes modificacions, ja estem en condicions de llançar la creació de la nova taula amb l'ordre **source** i la seva repoblació amb les dades relatives al genoma del ratolí amb l'ordre LOAD DATA.

```
DROP TABLE IF EXISTS 'refGene_mouse';

CREATE TABLE 'refGene_mouse' (
  'bin' smallint(5) unsigned NOT NULL,
  'name' varchar(255) NOT NULL,
  'chrom' varchar(255) NOT NULL,
  'strand' char(1) NOT NULL,
...
-----------------------------------------

mysql> source 'refGene_mouse.sql';

Query OK, 0 rows affected (0,00 sec)

mysql> LOAD DATA LOCAL INFILE 'refGene_mouse.txt'
    -> INTO TABLE refGene_mouse;

Query OK, 34904 rows affected (1,23 sec)
Records: 34904  Deleted: 0  Skipped: 0  Warnings: 0
```

Figura 69. Incorporar els gens de ratolí a la nostra base de dades.
Font: elaboració pròpia.

Comprovem que els registres emmagatzemats a la nova taula són correctes:

```
mysql> SELECT name2,name,chrom,strand,txStart,txEnd,exonCount
    -> FROM refGene_mouse ORDER BY name2 LIMIT 10;

+---------------+-------------+-------+--------+-----------+-----------+-----------+
| name2         | name        | chrom | strand | txStart   | txEnd     | exonCount |
+---------------+-------------+-------+--------+-----------+-----------+-----------+
| 0610005C13Rik | NR_038166   | chr7  | -      |  52823164 |  52830546 |         5 |
| 0610005C13Rik | NR_038165   | chr7  | -      |  52823164 |  52830546 |         4 |
| 0610007P14Rik | NM_021446   | chr12 | -      |  87156404 |  87165495 |         5 |
| 0610009B22Rik | NM_025319   | chr11 | -      |  51498886 |  51502136 |         2 |
| 0610009L18Rik | NR_038126   | chr11 | +      | 120209991 | 120212504 |         2 |
| 0610009O20Rik | NM_024179   | chr18 | +      |  38409902 |  38422283 |        13 |
| 0610010B08Rik | NM_001177543 | chr2 | -      | 175017505 | 175163713 |         6 |
| 0610010B08Rik | NM_001177543 | chr2 | -      | 174952492 | 175261278 |         6 |
| 0610010B08Rik | NM_001177543 | chr2 | +      | 175639522 | 175655901 |         5 |
| 0610010B08Rik | NM_001177543 | chr2 | +      | 175737073 | 175753460 |         5 |
+---------------+-------------+-------+--------+-----------+-----------+-----------+
10 rows in set (0,00 sec)
```

Figura 70. Visualitzar els primers transcrits del genoma del ratolí domèstic.
Font: elaboració pròpia.

Ara, si seleccionem aquells registres de les dues taules que pertanyen al mateix gen en ambdues espècies, podem construir un catàleg de gens homòlegs.

Podem dur a terme aquesta associació perquè SQL no distingeix entre majúscules o minúscules a l'hora de comparar la columna **name2**.

```
mysql> SELECT DISTINCT refGene.name2,refGene.chrom,refGene.strand,
    -> refGene.txStart,refGene.txEnd,refGene.exonCount,
    -> refGene_mouse.name2,refGene_mouse.chrom,
    -> refGene_mouse.strand,refGene_mouse.txStart,
    -> refGene_mouse.txEnd,refGene_mouse.exonCount
    -> FROM refGene JOIN refGene_mouse
    -> ON refGene.name2 = refGene_mouse.name2
    -> ORDER BY refGene.name2 ASC LIMIT 10;

+---------+-------+--------+----------+----------+-----------+---------+-------+--------+-----------+-----------+-----------+
| name2   | chrom | strand | txStart  | txEnd    | exonCount | name2   | chrom | strand | txStart   | txEnd     | exonCount |
+---------+-------+--------+----------+----------+-----------+---------+-------+--------+-----------+-----------+-----------+
| A1BG    | chr19 | -      | 58346805 | 58353499 |         8 | A1bg    | chr15 | -      |  60749143 |  60752825 |         7 |
| A1CF    | chr10 | -      | 50799408 | 50885675 |        13 | A1cf    | chr19 | +      |  31943250 |  32023896 |        12 |
| A1CF    | chr10 | -      | 50799408 | 50885675 |        14 | A1cf    | chr19 | +      |  31943250 |  32023896 |        12 |
| A1CF    | chr10 | -      | 50799408 | 50885675 |        15 | A1cf    | chr19 | +      |  31943250 |  32023896 |        12 |
| A2M     | chr12 | -      |  9067707 |  9116229 |        35 | A2m     | chr6  | +      | 121586190 | 121629256 |        36 |
| A2M     | chr12 | -      |  9067707 |  9116229 |        36 | A2m     | chr6  | +      | 121586190 | 121629256 |        36 |
| A2M     | chr12 | -      |  9067707 |  9116229 |        37 | A2m     | chr6  | +      | 121586190 | 121629256 |        36 |
| A3GALT2 | chr1  | -      | 33306765 | 33321098 |         5 | A3galt2 | chr4  | +      | 128436501 | 128446542 |         5 |
| A4GALT  | chr22 | -      | 42692111 | 42720910 |         3 | A4galt  | chr15 | -      |  83057151 |  83082161 |         3 |
| A4GALT  | chr22 | -      | 42692111 | 42720910 |         3 | A4galt  | chr15 | -      |  83057151 |  83082204 |         3 |
+---------+-------+--------+----------+----------+-----------+---------+-------+--------+-----------+-----------+-----------+
10 rows in set (5,14 sec)
```

Figura 71. Llistat de gens comuns entre el genoma humà i el genoma de ratolí.
Font: elaboració pròpia.

1. Bases de dades relacionals

1.15. *Triggers*, procediments i funcions

La majoria de bases de dades relacionals ofereixen la possibilitat d'emmagatzemar subprogrames. Es denominen funcions, procediments i *triggers* o disparadors, i són molt útils per automatitzar tasques i guardar instruccions SQL que s'han d'utilitzar freqüentment. Són objectes que contenen codi SQL, com breus *scripts* de codi SQL, que poden acceptar paràmetres i declarar variables.

S'assigna un nom al subprograma i s'executa perquè quedi emmagatzemat a la base de dades. Després el podem invocar o *cridar* pel seu nom perquè s'executi el codi emmagatzemat en els subprogrames.

- **Procediment** emmagatzemat. És un objecte que es crea amb la sentència `CREATE PROCEDURE` i s'invoca amb la sentència `CALL`. Els procediments poden acceptar paràmetres i no fan cap retorn, és a dir, no retornen cap
- **Funció** emmagatzemada. És un objecte que es crea amb la sentència `CREATE FUNCTION` i s'invoca amb la sentència **SELECT** o dins d'una expressió. Les **funcions** poden acceptar paràmetres i retornen sempre un valor. Aquest valor retornat pot ser un valor nul i en aquest cas es comportaria com un procediment.
- ***Trigger***. És un objecte que es crea amb la sentència `CREATE TRIGGER` i ha d'estar associat a una taula. Un *trigger* s'activa, es dispara, quan ocorre un esdeveniment d'inserció, actualització o esborrat, sobre la taula a la qual està associat.

Vegem-ne alguns exemples:

Procediments

1. Creem un procediment a la base de dades **cataleg** per comptar el nombre de gens diferents de la taula **refGene**, que anomenem **numGens**.

```
CREATE PROCEDURE numGens()
SELECT COUNT(distinct name2) FROM refGene;
```

Aquest procediment no té paràmetres () i mostra per pantalla el nombre de gens diferents que trobem al camp **name2** de la taula **refGene**. En executar el codi, el procediment s'emmagatzema a la base de dades com un objecte més, com les taules, i no s'executa la consulta `SELECT` que conté fins que l'anomenem.

Per anomenar el procediment **numGens** escrivim:

```
CALL numGens();
```

Si volem veure els procediments emmagatzemats a la base de dades escrivim:

```
SHOW PROCEDURE STATUS WHERE db = 'cataleg';
```

Si volem eliminar un procediment creat escrivim:

```
DROP PROCEDURE IF EXISTS numGens;
```

2. Ara utilizarem variables, el resultat del **SELECT** el guardarem en la variable local **gens**, que hem de declarar i de la qual hem de definir el tipus de dades.

Com que en el procediment hi ha sentències que acaben en «;», primer assignarem un nou delimitador. Escrivim:

```
DELIMITER //
```

Ara MySQL interpretarà que el final de les sentències SQL és el símbol //

Crearem un nou procediment anomenat **numGens2**.

Per declarar la variable **gens** necessitem escriure BEGIN abans de DECLARE i acabar amb END.

En declarar les variables locals cal especificar quin tipus de dada guardaran, INT, CHAR, VARCHAR, etc. En aquest cas és un INT.

```
DECLARE gens INT;
```

Per guardar el resultat del SELECT en la variable local **gens** fem:

```
INTO gens.
```

Per mostrar el contingut de la variable **gens** fem:

```
SELECT gens;
```

Veiem tot el codi per crear el nou procediment usant variables:

```
CREATE PROCEDURE numGens2()
BEGIN
DECLARE gens INT;
SELECT COUNT(distinct name2)
INTO gens
FROM refGene;
SELECT gens;
END //
```

Tornem a canviar el delimitador per poder fer servir «;» en finalitzar la sentència DELIMITER ;

Invoquem el procediment:

```
 CALL numGens2();
```

3. Ara passarem un paràmetre al procediment i l'utilitzarem en la condició WHERE de la consulta SELECT.

Creem un nou procediment anomenat **numTrans** que ens servirà per comptar el nombre de transcrits segons el tipus de transcrit que li passem com a paràmetre.

Igual que les variables locals, cal especificar als paràmetres quin tipus de dades esperen, INT, CHAR, VARCHAR... En aquest cas, CHAR(50).

El paràmetre el posem al costat del nom del procediment entre parèntesis numTrans(transcritType CHAR(50)).

Hi assignem una altra vegada un delimitador //:

```
DELIMITER //
```

Vegem el codi:

```
CREATE PROCEDURE numTrans ( transcritType CHAR(50))
BEGIN
DECLARE numT INT;
SELECT COUNT(*) INTO numT FROM refGene
WHERE name LIKE transcritType;
```

```
SELECT numT;
END //
```

Canviem una altra vegada el delimitador:

```
DELIMITER ;
```

Invoquem el procediment passant-li el paràmetre NR:

```
CALL numTrans('%NR%');
```

Ara invoquem el procediment passant-li el paràmetre NM:

```
CALL numTrans('%NM%');
```

En usar el comodí % ens assegurem que no perdem cap registre amb el contingut del paràmetre.

4. En comptes de declarar una variable local dins d'un procediment, també podem utilitzar un paràmetre com a variable de sortida. Indiquem que és un paràmetre de sortida amb la clàusula **OUT**. Per defecte, els paràmetres són només d'entrada, però també els podem indicar amb la clàusula **IN**.

Hi assignem un delimitador //:

```
DELIMITER //
```

Creem un nou procediment anomenat **numTrans2** que ens servirà també per comptar el nombre de transcrits segons el tipus de transcrit que li passem com a paràmetre, i amb un segon paràmetre que ens servirà per guardar el resultat de la consulta.

Vegem el codi:

```
CREATE PROCEDURE numTrans2(IN transcritType CHAR(50), OUT numT INT)
BEGIN
SELECT COUNT(*) INTO numT FROM refGene
WHERE name LIKE transcritType;
END //
```

Canviem el delimitador:

```
DELIMITER ;
```

Invoquem el procediment passant-li els dos paràmetres. Fem servir una variable definida per l'usuari amb @ trucada `@transcritos` per guardar el valor que retorna el **SELECT** del procediment:

```
CALL numTrans2('%NR%', @transcrits);
```

Fem un **SELECT** de la variable `@transcritos` que conté el nombre de transcrits:

```
SELECT @transcrits;
```

Funcions

Les funcions retornen un valor, així que, per anomenar una funció emmagatzemada, en comptes de fer CALL fem directament SELECT nombre_de_la_función i ens mostra el valor que retorna la funció.

Hi assignem un delimitador //:

```
DELIMITER //
```

Creem una nova funció anomenada **numTrans3** que ens servirà també per comptar el nombre de transcrits segons el tipus de transcrit que li passem com a paràmetre.

En les funcions hem d'indicar, després del nom i dels paràmetres, el tipus de dada que retorna la funció, en aquest cas un INT, i escrivim:

```
RETURNS INT
```

Al final de la funció li indiquem la variable que volem retornar, en aquest cas:

```
RETURN numT;
```

Vegem el codi íntegre de la funció:

```
CREATE FUNCTION numTrans3 (transcritType CHAR(50)) RETURNS INT
BEGIN
DECLARE numT INT;
SELECT COUNT(*) INTO numT FROM refGene
WHERE name LIKE transcritType;
RETURN numT;
END //
```

Canviem el delimitador:

```
DELIMITER ;
```

Invoquem la funció passant-li el paràmetre, en aquest cas els passem el paràmetre NR:

```
SELECT numTrans3('%NR%');
```

Si passem a la funció el paràmetre NM:

```
SELECT numTrans3('%NM%');
```

Si volem veure les funcions emmagatzemades a la base de dades escrivim la sentència:

```
SHOW FUNCTION STATUS WHERE db = 'cataleg';
```

Si volem eliminar una funció emmagatzemada, escrivim:

```
DROP FUNCTION IF EXISTS numTrans3;
```

Triggers

Un *trigger* és un objecte emmagatzemat a la base de dades que està associat amb una taula i que s'activa quan ocorre un esdeveniment sobre la taula.

Els esdeveniments que poden ocórrer sobre la taula són:

- **INSERT**. El *trigger* s'activa quan s'insereix una nova fila sobre la taula associada.
- **UPDATE**. El *trigger* s'activa quan s'actualitza una fila sobre la taula associada.
- **DELETE**. El *trigger* s'activa quan s'elimina una fila sobre la taula associada.

El *trigger* es pot activar o disparar abans (**BEFORE**) de l'esdeveniment o després (**AFTER**) de l'esdeveniment.

Com a exemples vam crear dos *triggers* associats a la taula *gens* de la nostra base de dades:

Un *trigger* amb el nom de **trig_check_gens_before_insert** que s'associa a la taula **gens**. S'activa abans d'una operació d'inserció. Si el nou valor del camp inici que es vol inserir és negatiu, es guarda com a 0. Si el nou valor del camp **final** que es vol inserir és menor que el valor del camp **inici**, es guarda el valor del camp **inici**.

Un *trigger* amb el nom de **trig_check_gens_before_update** que s'associa a la taula **gens**. S'activa abans d'una operació de modificació. Si el nou valor del camp **inici** que es vol modificar és negatiu, es guarda com a 0. Si el valor del camp **final** del registre que es vol modificar és menor que el nou valor que volem actualitzar del camp **inici** es guarda com a 1.

Creem el *trigger* **trig_check_gens_before_insert**

```
DELIMITER //
CREATE TRIGGER trig_check_gens_before_insert
BEFORE INSERT
ON gens FOR EACH ROW
BEGIN IF NEW.inici < 0 THEN SET NEW.inici = 0;
ELSEIF NEW.inici > NEW.final THEN SET NEW.final = NEW.inici;
END IF;
END//
```

En executar aquest codi de creació del *trigger,* **trig_check_gens_before_insert** queda emmagatzemat a la nostra base de dades, i només actuarà, s'activarà quan l'usuari executi una sentència d'inserció, per exemple:

Canviem el delimitador:

```
DELIMITER ;
```

Realitzem operacions d'inserció a la taula *gens* perquè es dispari el *trigger* **trig_check_gens_before_insert**:

```
 INSERT INTO gens (nom, cromosoma, cadena, inici, final, proteina,
especie) VALUES ('WASH7P', 'chrX', '+', -2527305,
2575270,'NR_033380', 'H. Sapiens');

 INSERT INTO gens (nom, cromosoma, cadena, inici, final, proteina,
especie) VALUES ('WASH7P2', 'chrX', '+', 252730599,
2575270,'NR_033381', 'H. Sapiens');
```

La variable composta **NEW** que utilitzem en el *trigger* emmagatzema tots els valors que inserim en cada operació **INSERT**. D'aquesta manera podem utilitzar-la en el *trigger* sense conèixer *a priori* quins valors s'hi inseriran.

En els nostres exemples la variable **NEW.inici** conté en el primer **INSERT** el valor -2527305 i en el segon **INSERT** la variable **NEW.inici** conté el valor 252730599 i la variable **NEW.final** conté el valor 2575270

En el primer **INSERT** es compleix `NEW.inici < 0`, aquesta condició dispara el *trigger* i en el camp **inici** es guarda el valor 0.

En el segon **INSERT** es compleix `NEW.inici > NEW.final`, aquesta condició dispara el *trigger* i en el camp **final** es guarda el valor `252730599`.

Ara creem el *trigger* **trig_check_gens_before_update**:

```
DELIMITER //
CREATE TRIGGER trig_check_gens_before_update
BEFORE UPDATE ON gens
FOR EACH ROW
BEGIN
IF NEW.inici < 0 THEN SET NEW.inici = 0;
ELSEIF NEW.inici > OLD.final THEN SET NEW.inici = 1;
END IF;
END //
```

En executar aquest codi de creació del *trigger*, **trig_check_gens_before_update** queda emmagatzemat a la nostra base de dades, i només actuarà, s'activarà quan l'usuari executi una sentència d'actualització, per exemple:

Canviem el delimitador:

```
DELIMITER ;
```

Realitzem operacions de modificació a la taula *gens* perquè es dispari el *trigger* **trig_check_gens_before_update**:

```
UPDATE gens SET inici = 228748314 WHERE nom = 'MYC';

UPDATE gens SET inici = -47643 WHERE nom = 'cbt';
```

En aquest cas, la variable composta **NEW** que utilitzem en el *trigger* emmagatzema el nou valor que volem actualitzar en la sentència **UPDATE**. En el primer **UPDATE** la variable `NEW.inici` conté el valor 228748314 i en el segon **UPDATE** la variable `NEW.inici` conté el valor `-47643`.

En canvi, la variable composta **OLD** emmagatzema tots els valors vells ja emmagatzemats a la taula del registre que es vol actualitzar. En el primer **UPDATE**, la variable `OLD.final` conté el valor emmagatzemat en el camp final del registre amb clau primària **MYC** en la taula **gens**, i en el segon **UPDATE**, la variable `OLD.final` conté el valor emmagatzemat en el camp final del registre amb clau primària **cbt**.

En el primer **UPDATE** es compleix `NEW.inici > OLD.final`, aquesta condició dispara el *trigger* i en el camp inici es guarda el valor 1.

En el segon **UPDATE** es compleix `NEW.inici < 0`, aquesta condició dispara el *trigger* i en el camp inici es guarda el valor 0.

En les operacions **DELETE** només s'utilitza la variable composta **OLD,** capaç d'emmagatzemar tots els valors del registre que es vol eliminar i accedir-hi, amb el format `OLD.nom_camp`.

2. Bases de dades NoSQL

2.1. Introducció

Les bases de dades relacionals són molt eficients i mantenen la integritat de les dades, però tenen limitacions per gestionar amb rapidesa grans volums d'informació.

Les bases de dades relacionals escalen de forma vertical. Per créixer necessiten que els servidors tinguin més capacitat.

MySQL és un dels SGBD més utilitzats per a projectes web, però les noves aplicacions web es caracteritzen per haver de gestionar un immens volum d'informació i gran quantitat de dades.

Per afrontar aquest nou repte de gestió de les dades, van aparèixer les bases de dades no relacionals, conegudes també com NoSQL o Not Only SQL, i anomenades així perquè no depenen únicament del llenguatge estructurat SQL.

Les bases de dades no relacionals poden escalar de forma horitzontal, permeten la distribució dels processos de treball i conjunts de dades en múltiples servidors. D'aquesta manera és possible que l'escalabilitat d'aquestes bases de dades sigui pràcticament il·limitada.

Les bases de dades NoSQL es classifiquen com de clau/valor, orientades a documents, grafs, o de famílies de columnes.

En aquest mòdul ens centrarem en les bases de dades no relacionals orientades a documents, concretament a documents en format JSON.

2. Bases de dades NoSQL

2.2. Fitxers JSON

JavaScript Object Notation (JSON) és un format de dades basat en text estàndard per representar dades estructurades que segueix la sintaxi d'objecte de JavaScript. Tot i que és molt semblant a la sintaxi d'objecte literal de JavaScript, pot ser utilitzat independentment de JavaScript.

Els fitxers per emmagatzemar dades amb format JSON cada vegada són més usats en la informàtica i també en l'àmbit de la bioinformàtica.

Figura 72. Exemple d'un document amb format JSON.
Font: elaboració pròpia.

En aquest format la forma de guardar les dades és semblant al model **clau/valor**, on la clau seria el nom del camp o atribut i a continuació tenim el valor.

Els fitxers JSON en realitat emmagatzemen una **col·lecció** de documents amb aquest format i cada document és la representació o la instància d'una entitat.

Si fem el símil amb la informació que emmagatzemem en una taula relacional, cada fila de la taula correspon a un document, i totes les files de la taula serien una col·lecció de documents.

JSON requereix usar cometes dobles per a les cadenes i els noms de propietats. Les cometes simples no són vàlides.

Una coma o dos punts mal ubicats poden produir que un arxiu JSON no funcioni. S'ha de ser curós per validar qualsevol dada que es vulgui utilitzar. És possible validar JSON utilitzant una aplicació com JSONLint.

Vegem com s'emmagatzema la informació que tenim guardada a la taula *genomes* en un fitxer JSON.

```
 {especie:"D. Melanogaster", nom:"Mosca de la fruita", descripcio:"També denominada del
 vinagre"} {especie:"H. Sapiens", nom:"Huma",descripcio:"Nostra pròpia especie"}
 {especie:"M. Musculus", nom:"Ratolí", descripcio:"Altre organisme model"}
```

En aquest cas tenim 3 documents que corresponen a les 3 files de la taula *genomes*. Cada document comença amb el símbol **{**, i finalitza amb el símbol **}**

En cada document es repeteix el nom del camp i a continuació el seu valor, separats pel símbol :

Cada combinació **camp/valor** se separa per comes.

No es descriuen els tipus de dades, si el valor és una cadena de caràcters s'utilitzen les cometes dobles " ", i si el valor és numèric no cal escriure'l entre cometes.

Una altra forma molt habitual de guardar la col·lecció de documents és dins d'un **array** amb els símbols **[ ]** i separant els diferents documents per comes. Es considera tota la col·lecció de documents com un sol objecte, ja que estan tots en un **array**.

```
 [{especie:"D. Melanogaster", nom:"Mosca de la
 fruita", descripcio:"També denominada del vinagre" }, {especie:"H.
 Sapiens", nom:"Huma", descripcio:"la nostra pròpia especie" },
 {especie:"M. Musculus", nombre:"Ratolí", descripcio:"Altre organisme
model" }]
```

Un camp pot guardar un **array** de valors:

```
 {dies:["dilluns", "dijous", "dissabte"]}
```

Un camp també pot guardar un altre **document JSON.** També s'anomena *document incrustat*.

```
 {direccio: {carrer: "Valencia", número: 334, codi: 08012}}
```

Un camp pot guardar un **array** de **documents JSON**.

```
 {amics:

 [{nom: "Pedro", edat: 34, telèfon: 666737211},

 {nom: "Soraya", edat: 31, telèfon: 666737212},

 {nom: "Arnau", edat: 29, telèfon: 666737213}

 ]}
```

A partir de la versió 8 de MySQL és possible treballar amb documents JSON a les taules SQL amb el tipus de dades JSON.

Podem emmagatzemar tot un document JSON en un camp de la taula i realitzar consultes amb l'ordre JSON_EXTRACT, actualitzacions amb JSON_REPLACE i eliminacions amb l'ordre JSON_REMOVE.

2. Bases de dades NoSQL

2.3. El SGDB MongoDB

Tot i que també és possible treballar amb les dades emmagatzemades en format JSON amb MySQL i altres SGBD relacionals com PostgreSQL, la millor forma de gestionar les dades emmagatzemades als fitxers JSON és utilitzant una base de dades NoSQL com MongoDB.

El SGBD MongoDB es va publicar l'any 2009 i permet gestionar bases de dades orientades a documents. Guarda els documents en BSON, que no és més que una implementació binària del format JSON.

MongoDB és la més popular de les bases de dades NoSQL. Bàsicament, retorna dades a JSON i incorpora els conceptes de col·leccions (en lloc de taules) i documents (en lloc de files), el seu API o llenguatge de consulta es coneix popularment com a MQL (MongoDB Query Language).

Perquè tinguem més clares les diferències entre el model relacional i MongoDB podem consultar la taula 4:

Taula 4. Comparativa entre el model relacional i MongoDB.

Model relacional	MongoDB
Database	Database
Table	Collection
Register	Document o BSON document
Columna	Field
Index	Index
Table joins	Embedded documents and linking
Primary key	Primary key
Specify any unique column or column combination as primary key	the primary key is automatically set to the_id field
Aggregation	Aggregation pipeline

Font: elaboració pròpia.

Bàsicament, la diferència més substancial és que mentre que en un SGBD relacional com MySQL tenim bases de dades, taules i columnes de les taules, a MongoDB i SGBD NoSQL basats en documents tenim també bases de dades, però en comptes de taules amb columnes hi tenim col·leccions de documents, i en cada document hi ha els noms dels camps en comptes de les columnes de les taules.

2. Bases de dades NoSQL

2.4. Començar a treballar amb el SGBD MongoDB

A la màquina virtual proporcionada per la UOC tenim instal·lat un SGBD MongoDB.

Per connectar-nos al servidor de MongoDB obrim un terminal, hi escrivim `mongosh` i ens sortirà el cursor >

```
student@ubuntuM0151:~$ mongosh
Current Mongosh Log ID: 66f3c526cf082f27c4964032
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.
3.1
Using MongoDB:          7.0.14
Using Mongosh:          2.3.1

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

------
   The server generated these startup warnings when booting
   2024-09-25T10:08:11.986+02:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. Se
e http://dochub.mongodb.org/core/prodnotes-filesystem
   2024-09-25T10:08:17.269+02:00: Access control is not enabled for the database. Read and write access to data and confi
guration is unrestricted
------
```

Figura 73. Exemple de connexió a MongoDB per terminal.
Font: elaboració pròpia.

Ja estem connectats al servidor amb el client de la línia d'ordres i ja podrem escriure les ordres i sentències per interactuar amb el servidor de MongoDB.

Per visualitzar les bases de dades creades:

```
show dbs
```

```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
```

Figura 74. Mostrar les bases de dades creades.
Font: elaboració pròpia.

Com que encara no hem creat cap base de dades, només ens mostra les bases de dades del sistema.

Per crear una nova base de dades buida fem servir l'ordre `use`.

Vam crear la base de dades **uoc**.

```
use uoc
```

L'ordre *use* serveix tant per crear una base de dades nova com per connectar-se a una base de dades existent. L'ordre intenta connectar-se a una base de dades existent i si no existeix la crea.

Aquesta és una característica general de MongoDB, és molt flexible i dona pocs errors. En altres sistemes de gestió de bases de dades com MySQL, en intentar connectar amb una base de dades inexistent saltaria un error. A MongoDB no salta error i es crea la nova base de dades. En realitat, no la crea, guarda espai per crear-hi col·leccions de documents. Si creem una nova base de dades amb l'ordre **use** i no creem col·leccions en ella, no queda emmagatzemada.

Veiem en aquesta seqüència d'ordres com consultem les bases de dades existents en el sistema amb `show dbs`, creem la base de dades **uoc** amb `use uoc`, i com quan tornem a consultar les bases de dades amb `show dbs` no apareix la base de dades

uoc.

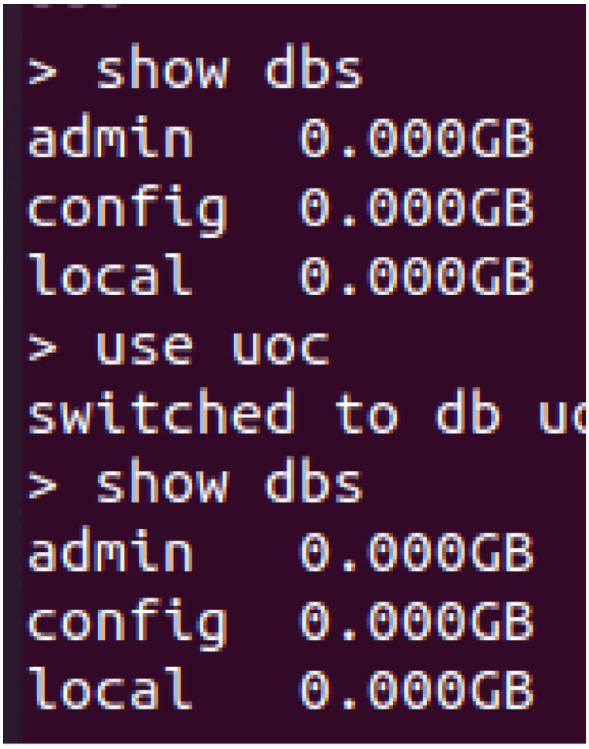


Figura 75. Utilitzar una base de dades.
Font: elaboració pròpia.

Perquè la nova base de dades quedi guardada al sistema cal que tingui col·leccions de documents.
Ara crearem una nova col·lecció buida a la base de dades **uoc** utilitzant la instrucció `createCollection()`.

```
> use uoc
switched to db uoc
> db.createCollection(chr1);
uncaught exception: ReferenceError: chr1 is not defined :
@(shell):1:1
> db.createCollection("chr1");
{ "ok" : 1 }
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
uoc     0.000GB
>
```

Figura 76. Crear una col·lecció de documents.
Font: elaboració pròpia.

Tornem a connectar-nos a la base de dades **uoc** amb `use uoc` i creem la col·lecció de documents buida anomenada **chr1**.

Si us hi fixeu, primer executem la instrucció o funció `db.createCollection(chr1);` però dona error perquè el paràmetre que espera la funció no està entre cometes simples: **"chr1"**.

En escriure el paràmetre entre cometes simples `db.createCollection("chr1");` la nova col·lecció anomenada **chr1** es crea correctament a la base de dades **uoc**, que ja no està buida, i en executar `show dbs` ja ens mostra la base de dades **uoc**.

Analitzarem la instrucció `db.createCollection("chr1");` i ens servirà per entendre com funcionen les instriccions a MongoDB. Com que ja som a la base de dades **uoc**, amb la part de la instrucció `db.....` es refereix a la base de dades que estem fent servir, i a continuació la instrucció `createCollection()` que en realitat és una funció que pot fer servir paràmetres entre parèntesis.

Com ja veurem, la majoria d'instruccions són funcions que poden fer servir o no paràmetres. Si la instrucció no necessita paràmetres deixarem els parèntesis buits **()**.

Anem a veure algunes **ordres útils** de MongoDB:

- `db.help()` Mostra l'ayuda per als mètodes de la base de dades.
- `db.<collection>.help()` Mostra l'ajuda per als mètodes de la col·lecció. La <collection> pot ser el nom d'una col·lecció ja creada o encara no creada.
- `Show db` Mostra la llista de totes les bases de dades del sistema.
- `use db` Canvia la base de dades actual a la base de dades.
- `show collections` Mostra la llista de totes les col·leccions de la base de dades actual.
- `show users` Mostra la llista de tots els usuaris de la base de dades actual.
- `show databases` Nou a partir de la versió 2.4. Mostra la llista de totes les bases de dades disponibles.
- `db.<collection>.find()` Mostra tots els documents de la col·lecció <collection>.
- `db.<collection>.find().pretty()` Mostra tots els documents de la col·lecció <collection> en un format JSON més llegible.

Per veure les col·leccions creades a la base de dades fem servir la instrucció següent:

```
show collections
```

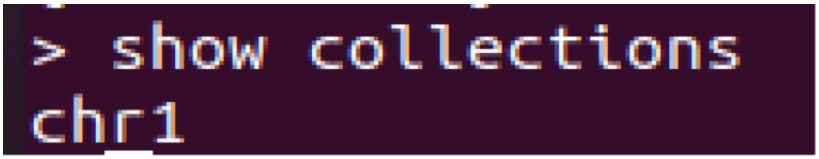


Figura 77. Mostrar les col·leccions de documents.
Font: elaboració pròpia.

2. Bases de dades NoSQL

2.5. Inserir documents

Com que tenim la col·lecció **chr1** buida hi inserirem documents:

Inserirem el següent document amb informació del gen **uc001aaa.3**, localitzat en el cromosoma 1:

```
{ "name": "uc001aaa.3", "chrom": "chr1", "strand": "+", "txStart":
11873, "txEnd": 14409, "cdsStart": 11873, "cdsEnd": 11873,
"exonCount": 3, "exonStarts": "11873,12612,13220,", "exonEnds":
"12227,12721,14409,", "proteinID": "", "alignID": "uc001aaa.3"}
```

Per inserir aquest document a la col·lecció **chr1** utilitzarem la instrucció `db.chr1.insert();` on s'indica amb `db` la base de dades a la qual estem connectats, `db.chr1...`, la col·lecció on inserim el nou document, la col·lecció **chr1,** i la funció insert() que espera com a paràmetre el document que vam inserir:

```
db.chr1.insert({ "name": "uc001aaa.3", "chrom": "chr1", "strand": "+",
"txStart": 11873, "txEnd": 14409, "cdsStart": 11873, "cdsEnd": 11873,
"exonCount": 3, "exonStarts": "11873,12612,13220,", "exonEnds":
"12227,12721,14409,", "proteinID": "", "alignID": "uc001aaa.3"});
```

Si la inserció s'ha realitzat correctament el sistema ens mostra el missatge següent:

```
WriteResult({ "nInserted" : 1 })
```

Ara hi inserirem dos documents més. Els documents JSON amb informació sobre els gens **uc010nxr.1** i **uc009vit.3**, també localitzats en el cromosoma 1.

```
db.chr1.insert({ "name": "uc010nxr.1", "chrom": "chr1", "strand": "+",
"txStart": 11873, "txEnd": 14409, "cdsStart": 11873, "cdsEnd": 11873,
"exonCount": 3, "exonStarts": "11873,12645,13220,", "exonEnds":
"12227,12697,14409,", "proteinID": "", "alignID": "uc010nxr.1"});


db.chr1.insert({ "name": "uc009vit.3", "chrom": "chr1", "strand": "-",
"txStart": 14361, "txEnd": 19759, "cdsStart": 14361, "cdsEnd": 14361,
"exonCount": 9, "exonStarts":
"14361,14969,15795,16606,16857,17232,17914,18267,18912,",
"exonEnds":
"14829,15038,15947,16765,17055,17742,18061,18366,19759,",
"proteinID": "", "alignID": "uc009vit.3"});
```

Ara eliminarem tota la col·lecció **chr1** de documents amb la instrucció `db.chr1.drop();`

I la tornem a crear:

```
db.createCollection("chr1");
```

per inserir els tres documents alhora amb la instrucció `insertMany();`

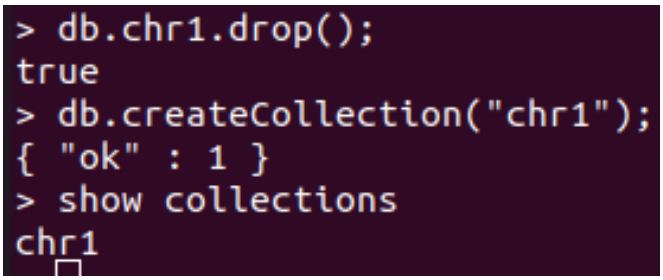


Figura 78. Eliminar una col·lecció de documents.
Font: elaboració pròpia.

A la part de fitxers JSON hem vist que moltes vegades trobem un fitxer amb una col·lecció de documents que estan tots en un *array*.

Els tres documents amb la informació dels gens dels exemples anteriors els tenim ara dins d'un *array*:

```
[{ "name": "uc001aaa.3", "chrom": "chr1", "strand": "+", "txStart":
11873, "txEnd": 14409, "cdsStart": 11873, "cdsEnd": 11873,
"exonCount": 3, "exonStarts": "11873,12612,13220,", "exonEnds":
"12227,12721,14409,", "proteinID": "", "alignID": "uc001aaa.3"},
{"name": "uc010nxr.1", "chrom": "chr1", "strand": "+", "txStart":
11873, "txEnd": 14409, "cdsStart": 11873, "cdsEnd": 11873,
"exonCount": 3, "exonStarts": "11873,12645,13220,", "exonEnds":
"12227,12697,14409,", "proteinID": "", "alignID": "uc010nxr.1"},
{"name": "uc009vit.3", "chrom": "chr1", "strand": "-", "txStart":
14361, "txEnd": 19759, "cdsStart": 14361, "cdsEnd": 14361,
"exonCount": 9, "exonStarts":
"14361,14969,15795,16606,16857,17232,17914,18267,18912,",
"exonEnds": "14829,15038,15947,16765,17055,17742,18061,18366,19759,",
"proteinID": "", "alignID": "uc009vit.3"}]
```

I els inserirem en la col·lecció **chr1** amb la instrucció `insertMany();`

```
db.chr1.insertMany([{ "name": "uc001aaa.3", "chrom": "chr1",
"strand": "+", "txStart": 11873, "txEnd": 14409, "cdsStart": 11873,
"cdsEnd": 11873, "exonCount": 3, "exonStarts": "11873,12612,13220,",
"exonEnds": "12227,12721,14409,", "proteinID": "", "alignID":
"uc001aaa.3"}, {"name": "uc010nxr.1", "chrom": "chr1", "strand": "+",
"txStart": 11873, "txEnd": 14409, "cdsStart": 11873, "cdsEnd": 11873,
"exonCount": 3, "exonStarts": "11873,12645,13220,", "exonEnds":
"12227,12697,14409,", "proteinID": "", "alignID": "uc010nxr.1"},
{"name": "uc009vit.3", "chrom": "chr1", "strand": "-", "txStart":
14361, "txEnd": 19759, "cdsStart": 14361, "cdsEnd": 14361,
"exonCount": 9, "exonStarts":
"14361,14969,15795,16606,16857,17232,17914,18267,18912,",
"exonEnds":
"14829,15038,15947,16765,17055,17742,18061,18366,19759,",
"proteinID": "", "alignID": "uc009vit.3"}]);
```

Ja tenim una altra vegada els tres documents a la col·lecció **chr1** de la base de dades **uoc,** i veurem algunes instruccions bàsiques per gestionar la informació amb MongoDB.

2. Bases de dades NoSQL

2.6. Buscar documents

Primer comptarem quants documents té la col·lecció chr1 amb la instrucció

```
db.chr1.find().count();
```

Aquesta instrucció conté dues funcions. La funció `find()` que és la funció que utilitzem per realitzar recerques en la col·lecció de documents. Seria la instrucció **SELECT** que hem vist a MySQL. Si no hi afegim paràmetres ens mostrarà tots els documents de la col·lecció, si hi afegim paràmetres, aquests seran els criteris de recerca o els filtres de la selecció. En aquest cas complementem la funció `find()` amb la funció `count()` per comptar els documents trobats. La funció `count()` no requereix paràmetres, però cal escriure els parèntesis igualment.

```
> db.chr1.find().count();
3
```

Figura 79. Comptar tots els documents d'una col·lecció.
Font: elaboració pròpia.

En executar la instrucció `db.chr1.find().count();` el sistema ens en retorna 3. La col·lecció **chr1** té 3 documents.

Utilitzarem la instrucció `db.chr1.find();` per veure tots els documents:

```
> db.chr1.find();
{ "_id" : ObjectId("64861afc32e0aa299185905c"), "name" : "uc001aaa
arts" : "11873,12612,13220,", "exonEnds" : "12227,12721,14409,", "
{ "_id" : ObjectId("64861df75d6ce7147d325b7e"), "name" : "uc010nxr
arts" : "11873,12645,13220,", "exonEnds" : "12227,12697,14409,", "
{ "_id" : ObjectId("64861e025d6ce7147d325b7f"), "name" : "uc009vit
arts" : "14361,14969,15795,16606,16857,17232,17914,18267,18912,",
>
```

Figura 80. Mostrar la informació de tots els documents d'una col·lecció.
Font: elaboració pròpia.

Per a cada document el sistema ens mostra el camp **_id** amb un valor generat aleatòriament per la funció del sistema `ObjectId();`

En el camp **_id** es guarda la clau primària del document. Aquesta clau primària és tan important per al sistema que si el document que inserim no té el camp **_id** el sistema el crea de forma automàtica. En els nostres exemples els tres documents inserits no tenen el camp **_id** i el sistema l'ha generat automàticament.

En un document hi pot haver molts camps i, tal com els mostra la funció `find();` , queden poc llegibles. Perquè el document es pugui llegir millor, la funció `find();` es pot complementar amb la funció `pretty();`

```
db.chr1.find().pretty();
```

Figura 81. Mostrar la informació de tots els documents d'una col·lecció amb un format més llegible.
Font: elaboració pròpia.

Com podem observar, la funció `pretty();` ens permet veure els valors de cada camp de cada document d'una forma molt més amigable.

Vegem ara alguns exemples d'utilització de la funció `find()` amb alguns paràmetres per filtrar els resultats.

Mostrarem només els gens de la nostra col·lecció que es troben en la cadena +:

```
db.chr1.find({"strand": "+"}).pretty();
```

Figura 82. Mostrar la informació dels gens de la cadena +.
Font: elaboració pròpia.

El paràmetre de la funció és la condició de selecció de la recerca `{"strand": "+"}`

Si no volem mostrar algun camp, com per exemple el camp `_id`, escrivim

```
db.chr1.find({"strand": "+"}, {"_id":0}).pretty();
```

```
> db.chr1.find({"strand": "+"}, {"_id":0}).pretty();
{
        "name" : "uc001aaa.3",
        "chrom" : "chr1",
        "strand" : "+",
        "txStart" : 11873,
        "txEnd" : 14409,
        "cdsStart" : 11873,
        "cdsEnd" : 11873,
        "exonCount" : 3,
        "exonStarts" : "11873,12612,13220,",
        "exonEnds" : "12227,12721,14409,",
        "proteinID" : "",
        "alignID" : "uc001aaa.3"
}
{
        "name" : "uc010nxr.1",
        "chrom" : "chr1",
        "strand" : "+",
        "txStart" : 11873,
        "txEnd" : 14409,
        "cdsStart" : 11873,
        "cdsEnd" : 11873,
        "exonCount" : 3,
        "exonStarts" : "11873,12645,13220,",
        "exonEnds" : "12227,12697,14409,",
        "proteinID" : "",
        "alignID" : "uc010nxr.1"
}
```

Figura 83. Mostrar la informació dels gens de la cadena + sense el camp `_id`.
Font: elaboració pròpia.

Si volem mostrar un sol camp dels documents seleccionats, com per exemple el camp **name**, escrivim

```
db.chr1.find({"strand": "+"},{"name":1, "_id":0 }).pretty();
```

```
> db.chr1.find({"strand": "+"},{"name":1, "_id":0 }).pretty();
{ "name" : "uc001aaa.3" }
{ "name" : "uc010nxr.1" }
```

Figura 84. Mostrar els gens de la cadena + sense mostrar el camp `_id` i mostrar el camp **name**.
Font: elaboració pròpia.

Les consultes amb la funció `find()` poden ser molt més elaborades, usant diversos camps com a condició de recerca, o també usant operadors especials com major que, menor que, etc.

Exemples d'operadors especials:

```
$gt, $gte, $lt, $lte, $ne, $in, $nin, $mod, $regex/$options,
```

```
$all, $size, $exists, $type, $not, $or, $nor, $elemMatch
```

Exemple de selecció de documents en què el camp **txStart** sigui major que 11873:

```
db.chr1.find({"txStart": {"$gt" : 11873 }}).pretty();
```

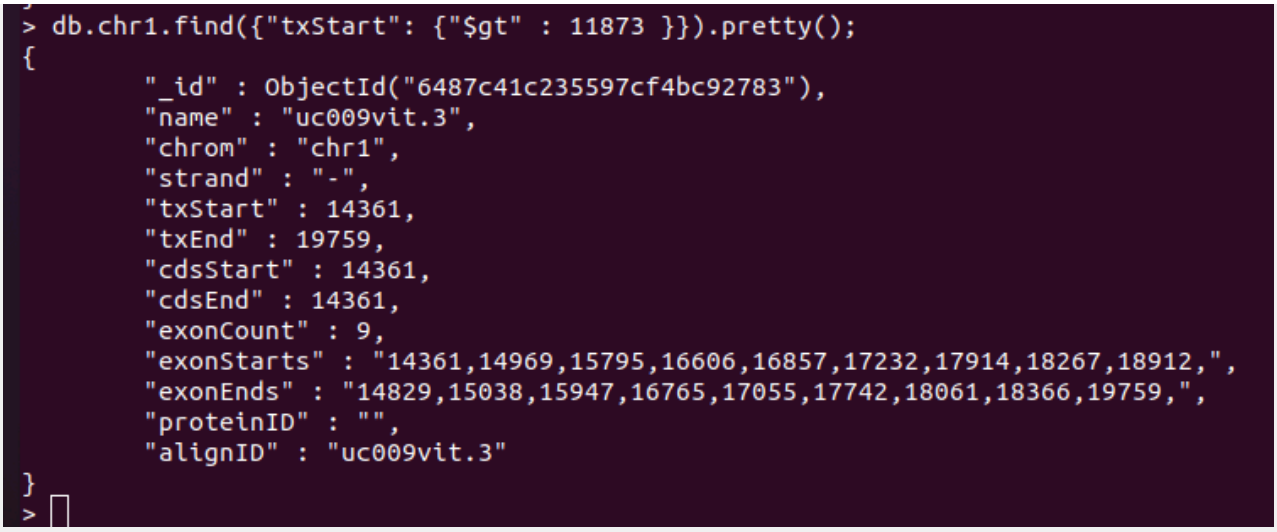


Figura 85. Mostrar els gens en què el camp txStar sigui més gran que 11873.
Font: elaboració pròpia.

O que compleixi dues condicions, que es trobin en la cadena + i que el camp **txStart** sigui major o igual a 11873.

```
db.chr1.find({"strand": "+", "txStart": {"$gte" : 11873
}}).pretty();
```

2. Bases de dades NoSQL

2.7. Modificar documents

Vegem ara com es pot actualitzar un document amb la funció `update()`.

Vam modificar la cadena del gen uc001aaa.3.

```
db.chr1.update({"name" : "uc001aaa.3"}, { "$set" : { "strand": "-" }});
```

I comprovem el canvi:

```
db.chr1.find({"name" : "uc001aaa.3"},{"name":1,"strand":1,"_id":0
}).pretty();
```

```
> db.chr1.update({"name" : "uc001aaa.3"}, { "$set" : { "strand": "-" }});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.chr1.find({"name" : "uc001aaa.3"},{"name":1,"strand":1,"_id":0 }).pretty();
{ "name" : "uc001aaa.3", "strand" : "-" }
>
```

Figura 86. Modificar la cadena del gen uc001aaa.3.
Font: elaboració pròpia.

L'operador `$set` modifica el valor de camp si aquest existeix; si no existeix el camp, l'incorpora al document o documents que coincideixin amb la selecció.

L'operador `$unset` elimina el camp del document o documents que coincideixin amb la selecció.

També és possible afegir nous elements a un camp *array* amb l'operador `$push` i eliminar elements de l'*arrayt* amb els operadors `$pull`, `$pullAll`, `$pop`.

Els operadors de l'ordre update són els següents:

```
$set, $unset, $inc, $push, $pushAll, $pull, $pullAll, $pop,
```

```
$addToSet, $rename, $bit, $ positional operator
```

2. Bases de dades NoSQL

2.8. Eliminar documents

Vegem ara com podem eliminar un document de la col·lecció amb la funció `remove()`.

Eliminem de la col·lecció el document amb el nom de gen **uc001aaa.3**

```
db.chr1.remove( { "name" : "uc001aaa.3"});
```

I comprovem si el document s'ha eliminat:

```
db.chr1.find().pretty();
```

```
> db.chr1.remove( { "name" : "uc001aaa.3"});
WriteResult({ "nRemoved" : 1 })
> db.chr1.find().pretty();
{
        "_id" : ObjectId("6487c410235597cf4bc92782"),
        "name" : "uc010nxr.1",
        "chrom" : "chr1",
        "strand" : "+",
        "txStart" : 11873,
        "txEnd" : 14409,
        "cdsStart" : 11873,
        "cdsEnd" : 11873,
        "exonCount" : 3,
        "exonStarts" : "11873,12645,13220,",
        "exonEnds" : "12227,12697,14409,",
        "proteinID" : "",
        "alignID" : "uc010nxr.1"
}
{
        "_id" : ObjectId("6487c41c235597cf4bc92783"),
        "name" : "uc009vit.3",
        "chrom" : "chr1",
        "strand" : "-",
        "txStart" : 14361,
        "txEnd" : 19759,
        "cdsStart" : 14361,
        "cdsEnd" : 14361,
        "exonCount" : 9,
        "exonStarts" : "14361,14969,15795,16606,16857,17232,17914,18267,18912,",
        "exonEnds" : "14829,15038,15947,16765,17055,17742,18061,18366,19759,",
        "proteinID" : "",
        "alignID" : "uc009vit.3"
}
```

Figura 87. Eliminar el gen uc001aaa.3.
Font: elaboració pròpia.

Ara eliminarem tota la col·lecció amb l'ordre `drop`

```
db.dropDatabase();
```

2. Bases de dades NoSQL

2.9. Importar fitxers JSON a MongoDB

Ara treballarem important un fitxer en format JSON que ens proporciona el navegador genòmic UCSC.

Descarreguem el fitxer JSON del següent enllaç:
https://api.genome.ucsc.edu/getData/track?genome=hg19;track=knownGene;chrom=chr1

Aquest fitxer conté la informació de tots els gens coneguts del cromosoma 1.

Baixem el fitxer i el guardem amb el nom hg19chr1.json

Per importar el fitxer a MongoDB obrim un nou terminal a la carpeta de la màquina virtual on hem guardat el fitxer i escrivim

```
mongoimport --db hg19 --collection chr1 --drop --file hg19chr1.json
```

Analitzarem aquesta instrucció:

L'ordre `mongoimport` s'executa **fora** del client (mongo) i té diverses opcions o paràmetres

- `--db` indica la base de dades. Si existeix la utilitza per crear la col·lecció de documents, si no existeix la crea.
- `--collection db` indica la col·lecció. Si existeix la utilitza per inserir-hi els documents del fitxer que importem, si no existeix la crea.
- `--drop elimina` els documents previs de la col·lecció, si existeix.
- `--file` indica el fitxer que importarem. Si els documents estiguessin dins d'un *array* hem d'afegir l'opció `--jsonArray`

Si ens connectem ara al servidor de MongoDB amb el client mongo i mirem les bases de dades amb `show dbs`, veiem com s'ha creat la base de dades **hg19.**

Si ens connectem a la base de dades **hg19** amb `use hg19` i veiem les col·leccions amb `show collections` podem veure la col·lecció **chr1** que hem creat i ja podem treballar amb ella.

2. Bases de dades NoSQL

2.10. Buscar en un *array* de documents

Si ens fixem en l'estructura del fitxer JSON importat, només conté un document JSON. I en un dels camps, l'anomenat **knownGene**, conté un *array* de documents JSON, i cada document de l'*array* conté informació dels gens del cromosoma 1: nom, cadena, cromosoma, etc.

Això ens obliga a conèixer com es treballa amb els continguts dels *arrays* si volem gestionar correctament aquesta informació.

Hem de fer servir el `dot.notation`.

Per exemple, mostrarem la informació del gen `uc009vis.3`.

Aquesta és la instrucció:

```
 db.chr1.find({ "knownGene.name": "uc009vis.3"}, {"knownGene.$":
1}).pretty();
```

Figura 88. Mostrar la informació del gen uc009vis.3.
Font: elaboració pròpia.

Veiem com per referir-nos al nom del gen escrivim `"knownGene.name":` és a dir, el nom del camp del document que és un *array* de documents JSON, el `knownGene`, punt i a continuació el nom del camp dels documents que es troben *a* l'*array* `name`. Amb `knownGene.$": 1` indiquem que ens mostri tots els camps dels documents trobats a l'*array* `knownGene`.

El `dot.notation` també ens serveix per referir-nos a camps de documents embeguts.

2. Bases de dades NoSQL

2.11. Agregacions a MongoDB

Per treballar a fons amb els elements d'un *array* és millor utilitzar el `framework aggregation` que ens permet MongoDB, moltes opcions i realitzar tubs *pipelines* per gestionar la informació emmagatzemada.

Operacions d'agregacions

Les operacions d'agregació són eines de MQL que ens ajuden a processar documents i a retornar resultats calculats. Les operacions d'agregació s'utilitzen majoritàriament per:

- Agrupar valors de diversos documents.
- Processar i operar per al retorn de resultats.
- Analitzar canvis de dades al llarg del temps.

Tubs i transformacions

Per realitzar el processament de documents, MongoDB es basa en el patró de filtre de tubs, utilitzat comunament en arquitectures de programari. Aquest patró consta d'una o més etapes, on cada etapa realitza una operació amb les dades d'entrada, i la sortida o resultat l'entrega a la següent etapa per al seu processament.

Per aplicar aquest patró, MongoDB utilitza una sèrie d'operadors ja definits per poder processar documents.

Els operadors més utilitzats en tubs:

- Filtratge de documents amb criteris: `$match`.
- Ordre de documents: `$sort`.
- Selecció de camps en específic: `$project`.
- Agrupació de documents: `$group`.
- Treure els elements d'un *array*: `$unwind`.

Caldria un altre curs per aprofundir en totes les opcions que ens ofereix MongoDB.

Vegem un exemple d'una operació d'agregació. La funció `aggregate()`.

Entre moltes altres opcions, `aggregate` ens permet treure els elements d'un *array* amb l'operador `$unwind`, fer agrupacions amb `$group` i comptar amb `$sum`.

Anem a comptar quants gens conté cada fil de l'ADN del cromosoma 1.

```
db.getCollection('chr1').aggregate([{$unwind:"$knownGene"},{$group:{_id:"$knownGene.strand",cantidad:{$sum:1}}}])
```

```
> db.getCollection('chr1').aggregate([{$unwind:"$knownGene"},{$group:{_id:"$knownGene.strand",cantidad:{$sum:1}}}]);
{ "_id" : "-", "cantidad" : 3894 }
{ "_id" : "+", "cantidad" : 4073 }
>
>
>
```

Figura 89. Mostrar quants gens conté cada fil de l'ADN del cromosoma 1.
Font: elaboració pròpia.

Resum

A nivell genòmic, habitualment treballem amb milers de registres durant una anàlisi bioinformàtica convencional. Malgrat que les eines d'anàlisi de fitxers de text basades en l'ús d'ordres del terminal de GNU/Linux ens permeten accedir fàcilment a les nostres dades, els sistemes de gestió de bases de dades com MySQL o MongoDB resulten el mitjà ideal per gestionar grans volums de dades de forma eficient i ràpida.

En aquest mòdul us hem mostrat un gran ventall d'ordres basades en el llenguatge SQL per consultar les taules de les nostres bases de dades relacionals i algunes ordres específiques de MongoDB per gestionar la informació d'una base de dades no relacional NoSQL orientada a documents.

Juntament amb aquest inventari d'instruccions, hem après també a modelar correctament les entitats del problema real que desitgem aproximar mitjançant estratègies bioinformàtiques.

Activitats

1. Portem a la pràctica sobre el teu propi SGBD MySQL els exemples mostrats durant aquest mòdul. Intentem enriquir cada base de dades amb noves taules que modelin entitats o relacions que no apareixien originalment en els casos estudiats. Ampliem el conjunt d'atributs de les taules per descriure amb més precisió les instàncies reals que es mostren al llarg del text.

2. Dissenyem una base de dades per emmagatzemar informació sobre un entorn complex natural que ens agradaria modelar: un ecosistema, el cos humà, la cèl·lula o l'univers. Reflexionem sobre les entitats, els seus atributs i les relacions entre aquestes, que són necessàries en cada cas per especificar aquest model. Repetim l'exercici amb un entorn generat per l'ésser humà (per exemple, un automòbil).

3. Implementem una base de dades a MySQL que permeti portar el registre de totes les activitats d'un servidor web genèric que rep peticions per executar una sèrie de serveis: pàgines més visitades, tipus de serveis sol·licitat, volum de dades, temps de resposta, usuaris, nombre de pàgines consultades per client, adreça IP i nom de la màquina, país de procedència, etc.

4. Ampliem el nostre catàleg de gens pensant en futures ampliacions. Podem afegir-hi una entitat PROTEÏNES per emmagatzemar informació estructural o sobre dominis funcionals. També seria interessant incorporar-hi una entitat CROMOSOMES, relacionada amb la taula GENOMAS, on guardar altres característiques, com el nombre de bases o el contingut en G + C.

5. Importem alguns fitxers JSON que ens proporciona UCSC Genome Browser https://genome.ucsc.edu/goldenPath/help/api.html#REST a MongoDB creant per a cadascun d'ells una base de dades nova i una nova col·lecció. Realitzem alguna consulta simple a les col·leccions i alguna consulta en un document situat en un camp *array* de documents.

Exercicis d'autoavaluació

1. Definiu el model entitat-relació en el disseny de bases de dades.
2. Descriviu les diferències entre bases de dades i gestors de base de dades.
3. Descriviu què és una taula en el model relacional.
4. Què són les claus primàries?
5. Què són les claus foranes?
6. Definiu què és una relació 1:N.
7. Recordeu quin símbol heu d'emprar sempre al final de cada ordre SQL.
8. Esmenteu cinc ordres essencials del llenguatge SQL.
9. Descriviu l'ordre de SQL per explorar el contingut de les taules.
10. Quin tipus d'operacions es realitzen sobre registres agrupats?
11. Diferencieu entre les ordres DISTINCT i LIMIT.
12. Enumereu els tres tipus d'unions a realitzar entre dues taules.
13. Definiu la utilitat d'una subconsulta.
14. Quines dues ordres són útils per esborrar taules? En què es diferencien?
15. Indiqueu la instrucció de SQL per executar un fitxer d'ordres.
16. Indiqueu la instrucció de SQL per carregar un fitxer de dades.
17. Quin format compleixen els fitxers de text per emprar-se d'aquesta manera?
18. Indiqueu el programa de GNU/Linux que copia una base de dades en un fitxer.
19. Com es pot iniciar el client de MySQL des de la línia d'ordres?
20. Com es concedeixen autoritzacions a un usuari sobre una base de dades MySQL?
21. Com es mostra un document amb un format més llegible?
22. Com es crea una nova col·lecció de documents a MongoDB?
23. Com es compten els documents d'una col·lecció a MongoDB?
24. Què és la sintaxi dot.notation a MongoDB?

Solucionari

1. El model entitat-relació es basa en l'ús de dos tipus d'elements per modelar un entorn real. Les entitats modelen cada classe d'elements de la realitat. Les relacions modelen les associacions entre les instàncies de cada entitat en aquest entorn.

2. Una base de dades és un conjunt d'informacions organitzades per fomentar un accés eficient a aquestes; un gestor de base de dades és precisament el programa que implementa el manteniment permanent de la base de dades, i que, a més, ofereix mecanismes per accedir-hi.

3. Una taula és una estructura que agrupa una col·lecció d'elements (instàncies) de la mateixa classe. Generalment, una taula modela una entitat juntament amb els seus atributs, tot i que pot ser necessària també per implementar algunes relacions entre entitats dins de la base de dades.

4. La clau primària d'una taula és l'atribut que identifica de forma unívoca cada instància o element en el seu interior. Per això, el valor d'aquest atribut no es pot repetir entre instàncies diferents.

5. La clau forana d'una taula és la clau primària d'una altra taula, assegurant la integritat del model, atès que cada instància a la primera taula haurà d'existir també a la segona.

6. Una relació 1:N entre dues entitats indica que cada instància de la primera entitat pot associar-se amb *N* instància a la segona taula. Pot implementar-se en la segona taula directament amb un atribut que prengui per valor algun dels valors del rang correcte per a la primera.

7. Qualsevol ordre que introduïm en l'intèrpret de MySQL ha d'acabar obligatòriament amb el símbol ;, per ser executada correctament.

8. Per exemple: `CREATE DATABASE`, `CREATE TABLE`, `SELECT`, `LOAD DATA` i `GRANT`.

9. L'ordre `SELECT... FROM... WHERE` permet realitzar una consulta sobre els registres de les taules que compleixen certes condicions.

10. Quan s'agrupen instàncies amb `GROUP BY`, és possible calcular mitjanes aritmètiques, mínims, màxims o comptes de totals (`AVG`, `MIN`, `MAX`, `COUNT`).

11. La clàusula DISTINCT serveix per eliminar valors repetits. La clàusula LIMIT és útil per mostrar només les primeres instàncies d'una taula.

12. A l'hora de comparar dues taules amb l'ordre `JOIN` emprant un atribut en comú, podem buscar les parelles de valors presents en ambdues taules o aquelles que només apareixen en una d'elles (`LEFT` o `RIGHT`).

13. Una subconsulta permet generar un grup de resultats en forma de taula temporal. Aquesta taula auxiliar podrà ser interrogada, al seu torn, per la consulta principal que alberga la subconsulta en el seu interior.

14. L'ordre `DROP TABLE` elimina la taula completament (definició i contingut). L'ordre `DELETE`, en canvi, elimina únicament determinats registres d'acord amb una condició.

15. La instrucció `SOURCE`.

16. La instrucció `LOAD DATA`.

17. El fitxer de text ha d'estar tabulat en columnes, que corresponen als atributs de les taules.

18. El programa **mysqldump** realitza el bolcat de la base de dades.

19. L'ordre seria: `mysql -o usuari -p`

20. Amb la instrucció `GRANT`

21. Amb l'ordre o funció `pretty()`

22. Amb la funció `db.createCollection("nomColecció");`

23. Amb la funció `count();`

24. És la forma d'identificar un camp situat en un document situat en camp *array* de documents de la forma
    `nom_camp_array.nom_ camp_ document`

25. És un document situat en un camp d'un document JSON.

Bibliografia

Conrad Bessant, Ian Shadforth i Darren Oakley (2009). *Building Bioinformatics Solutions: with Perl, R and MySQL*. Oxford University Press. ISBN: 0199230234.

Edgar F. Codd (1970). A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13, p. 377-387.

MySQL AB (2006). *Manual de referencia de MySQL 8.0*. http://dev.mysql.com

Paul DuBois (2008). *MySQL* (4th Edition). Addison-Wesley Professional. ISBN: 0672329387.

Peter Pin-Shan Chen (1976). The Entity-relationship Model–Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1, p. 9-36.

Steven Haddock i Casey Dunn (2011). *Practical Computing for Biologists*. Sinauer Associates. ISBN: 978-0-87893-391-4.

Vince Buffalo (2015). *Bioinformatics Data Skills.* O'Reilly Media. ISBN: 978-1-449-36737-4.

Webgrafia

Anaya Multimedia. Guía Práctica MySQL 5.1/Capítulo 11: Procedimientos almacenados. https://enreas.fandom.com/wiki/Gu%C3%ADa_Pr%C3%A1ctica_MySQL_5.1/Cap%C3%ADtulo_11:_Procedimientos_almacenados

COMANDOS MYSQL. Convertir consulta MySQL a JSON. https://thedevelopmentstages.com/convertir-consulta-mysql-a-json/

DelftStack. Cómo declarar y usar las variables en MySQL. https://www.delftstack.com/es/howto/mysql/mysql-declare-variable/

Guebs. MySQL 5.0 Reference Manual. Traducción. https://manuales.guebs.com/mysql-5.0/

Guebs. MySQL 5.0 Reference Manual. 19.2.1. CREATE PROCEDURE y CREATE FUNCTION. https://manuales.guebs.com/mysql-5.0/stored-procedures.html#create-procedure

JSONLint – The JSON Validator. https://jsonlint.com/

MongoDB. https://www.mongodb.com/

MongoDB. MongoDB CRUD Operations. https://www.mongodb.com/docs/manual/crud/

MongoDB Manual. Aggregation Operations. https://www.mongodb.com/docs/manual/aggregation/

MySQL. https://dev.mysql.com/

MySQL 8.0 Reference Manual. https://dev.mysql.com/doc/refman/8.0/en/

MySQL 8.0 Reference Manual. 13.1.20.5 FOREIGN KEY Constraints. https://dev.mysql.com/doc/refman/8.0/en/create-table-foreign-keys.html

MySQL 8.0 Reference Manual. 13.1.17 CREATE PROCEDURE and CREATE FUNCTION Statements. https://dev.mysql.com/doc/refman/8.0/en/create-procedure.html

MySQLTUTORIAL.MySQL Stored Procedures. https://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx

MySQL 8.0 Reference Manual. 25.3.1 Trigger Syntax and Examples. https://dev.mysql.com/doc/refman/8.0/en/trigger-syntax.html

MySQL 8.0 Reference Manual. 11.5 The JSON Data Type. https://dev.mysql.com/doc/refman/8.0/en/json.html

Universidad de Sevilla. Introducción a PL/SQL en MySQL. https://dam.org.es/introduccion-a-pl-sql-en-mysql/