Importar: meter datos de sakila en la db
Exportar: exportar datos de la base a un archivo.

QUERIES:
Query 1 : buscar una película (IRON MOON) por su título y devuelva su descripción.
Query 2: número de películas clasificadas como R.
Query 3: media de número de actores que participan en películas que duran entre 90 y 120 minutos.
Query 4: top 10 de los clientes que más alquileres efectúan

Rendimiento (server)	create & insert	q1	q2	q3	q4	export
MySQL	2813 ms	0,7 ms	1 ms	2 ms	6 ms	248 ms
MongoDB	540 ms	0 ms	0 ms	1 ms	-	679 ms
CouchDB	-	-	-	-	-	-
Neo4j	30 min aprox.	4 ms	4 ms	9 ms	16 ms	8343 ms
SQLite	2486 ms (No es necesario)	1 ms	1 ms	1 ms	9 ms	84 ms (No es necesario)

Rendimiento (local)	create & insert	q1	q2	q3	q4	export
MySQL	1876 ms	0,5 ms	1,3 ms	2,5 ms	8 ms	189 ms
MongoDB	292 ms	2 ms	1 ms	3 ms	-	738 ms
CouchDB (10000 instancias)	166 000 ms (2 min 46 s)	800 ms	1100 s	900 ms	867 ms	2331ms
Neo4j	6 min	2 ms	2 ms	6 ms	10 ms	5587 ms
SQLite	3757 ms (No es necesario)	2 ms	2 ms	2 ms	16 ms	272 ms (No es necesario)

El tiempo se ha calculado después de varias ejecuciones; es una media, no el mejor ni el peor valor.

Usabilidad	create & insert	q1	q2	q3	q4	export
MySQL	Fácil	Sencillo y fácil de leer	Sencillo y fácil de leer	Muy difícil y difícil de leer	Difícil y dificultad media en la lectura	Fácil
MongoDB	Fácil (a partir de comandos sencillos)	Sencillo	Sencillo y fácil de leer	Muy difícil, complicado de leer	No es posible	Fácil (a partir de comandos sencillos)
CouchDB	Fácil (mediante el uso de una librería python)	Sencillo y fácil de leer	Dificultad media, menos fácil de leer	Dificultad media, menos fácil de leer	Fácil y bastante fácil de leer	Complejo, se debe escribir código propio
Neo4j	Fácil (a partir de un .cypher)	Sencillo y fácil de leer	Sencillo y fácil de leer	Dificultad media, fácil de leer	Sencillo y fácil de leer	Complejo la primera vez (configuración)
SQLite	Fácil	Sencillo y fácil de leer	Sencillo y fácil de leer	Muy difícil y difícil de leer	Difícil y dificultad media en la lectura	Fácil

MYSQL

Importar: **time ./crear.sh**

Exportar: **time sudo mysqldump -u root sakila > sakila.sql**

Query 1:

```
SELECT description FROM sakila.film WHERE title="IRON MOON";
```

Query 2:

```
SELECT COUNT(*) FROM sakila.film WHERE rating='R';
```

Query 3:

```
SELECT AVG(amounts.actors_amount) AS actors_amount_avg
FROM (SELECT COUNT(a.actor_id) AS actors_amount
        FROM sakila.film f, sakila.film_actor a
        WHERE f.film_id=a.film_id AND f.length>=90 AND f.length<=120
        GROUP BY f.film_id) AS amounts;
```

Query 4:

```
SELECT first_name, last_name, COUNT(rental_id) as rentals_amount
FROM sakila.customer c, sakila.rental r
WHERE c.customer_id=r.customer_id
GROUP BY c.customer_id
ORDER BY rentals_amount DESC
LIMIT 10;
```

MONGODB

Importar: **time sh imports.sh**

Exportar: **time sh exports.sh**

Queries:

1- db.films.find({Title:"IRON MOON"}, {Description:1, _id:0})
2- db.films.count({Rating:"R"})
3- db.films.aggregate({$match: { $and: [{ Length: { $lt: "90" } }, { Length: { $lt: "120" } }] }}, {$group:{_id:null, media:{$avg:"$Length"}}})
4- No es posible

Calcular tiempo de ejecucion:

1- db.films.find({Title:"IRON MOON"}, {Description:1, _id:0}).explain("executionStats")
2- db.films.explain("executionStats").count({Rating:"R"})
3- db.films.explain("executionStats").aggregate({$match: { $and: [{ Length: { $lt: "90" } }, { Length: { $lt: "120" } }] }}, {$group:{_id:null, media:{$avg:"$Length"}}})
4- No es posible

COUCHDB

Importar: **time python createImport.py**
Exportar: **time couchdb-dump.sh** (NOTA: Script descargado de un repositorio github)

QUERIES PARA CALCULAR RENDIMIENTO:

Query 1 : buscar un usuario (Wilcox Randolph) por su título y que devuelva su descripción.

View: (javascript)

```
map: function(doc){
        if(doc.name==”Wilcox Randolph”){
                emit(doc._id, doc.about);
        }
}
```

Query: (python)

```
for item in db.view(‘queries/query1’): print(item.value)
```

Query 2: número de usuarios cuya fruta favorita es ‘apple’.

View: (javascript)

```
map: function(doc){
        if(doc.favoriteFruit==”apple”){
                emit(doc.favoriteFruit,1);
        }
}
reduce: function(keys,values,rereduce){
        if(rereduce){
                return sum(values);
        }else{
                return sum(values);
        }
}
```

Query: (python)

```
for item in db.view(‘queries/query2’, group=True): print(item.value)
```

Query 3: media de edad de usuarios que tienen un balance entre $1500 y $2000.

View: (javascript)

```
map: function(doc){
        if(doc.balance>"$1,500.00" && doc.balance<"$2,000.00"){
                emit(1, doc.age);
        }
}
reduce: function(keys,values,rereduce){
        if(rereduce){
                return (sum(values)/values.length);
        }else{
                return (sum(values)/values.length);
        }
}
```

Query: (python)

```
for item in db.view('queries/query3', group=True): print(item.value)
```

Query 4: top 10 de los usuarios que tienen mayor balance

View: (javascript)

```
map: function(doc){
        emit(doc.balance, doc.name);
}
```

Query: (python)

```
for item in db.view(doc1/query4', limit=10, descending=True):
print(item.key,item.value)
```

QUERIES PARA COMPROBAR USABILIDAD:

Query 1:

View: (javascript)

```
map: function(doc){
        if(doc.Title==”IRON MOON”){
                emit(doc._id, doc.Description);
        }
}
```

Query: (python)

```
for item in db.view(“doc1/query1’): print(item.value)
```

Query 2:

View: (javascript)

```
map: function(doc){
        if(doc.Rating==”R”){
                emit(doc.Rating,1);
        }
}
reduce: function(keys,values,rereduce){
        if(rereduce){
                return sum(values);
        }else{
                return sum(values);
        }
}
```

Query: (python)

```
for item in db.view(‘doc1/query2’, group=True): print(item.value)
```

Query 3:

View: (javascript)

```
map: function(doc){
        if(doc.Length>90 && doc.Length<120){
                emit(1, doc.Actors.length);
        }
}
reduce: function(keys,values,rereduce){
        if(rereduce){
                return (sum(values)/values.length);
        }else{
                return (sum(values)/values.length);
        }
}
```

Query: (python)

```
for item in db.view('doc1/query3', group=True): print(item.value)
```

Query 4:

View: (javascript)

```
map: function(doc){
        emit(doc.FirstName, doc.Rentals.length);
}
```

Query: (python)

```
for item in db.view('doc1/query4', limit=10, descending=True):
        print(item.key,item.value)
```

ventajas:
Las bases de datos son más fáciles de replicar, una vez se aprende a cómo ejecutar una query resulta bastante sencillo.

desventajas:
Mayor desorden en la base de datos (dependencia en usar vistas), curva de aprendizaje más grande al ser tan diferente a lo empleado anteriormente, falta de documentación clara para ciertas tareas. No dispone de una forma para exportar una base de datos a un fichero '.JSON' que después sea compatible con el formato deseado por CouchDB. (Se debe crear tu propio código o emplear el de otro usuario).

NEO4J

Importar: **cypher-shell -f all.cypher**

Exportar (desde cypher-shel): **CALL apoc.export.cypher.all("sakila.cypher")**

query 1:

```
MATCH (iron:Film {title: "IRON MOON"}) RETURN iron.description
```

```
MATCH (iron:Film) WHERE iron.title="IRON MOON" RETURN iron.description
```

query 2:

```
MATCH (r:Film {rating: "R"}) RETURN count(r)
```

query 3:

```
MATCH (a:Actor)-[:FILM_ACTOR]->(f:Film) WHERE f.length<=120 AND f.length<=90
with f,count(a) as amount return avg(amount)
```

query 4:

```
MATCH p=(c:Customer)--(r:Rental) return c.firstName, c.lastName,count(r) as amount
order by amount desc limit 10
```

para evitar el warning (no afecta al rendimiento):

```
MATCH p=(c:Customer)--(r:Rental) return c.firstName, c.lastName,count(r) as amount
order by count(r) desc limit 10
```

ventajas:
Facilidad de uso.

desventajas:
No está pensado para superar a alternativas como mysql en este tipo de tareas.

SQLITE

Importar:
time ./sakila_generator.sh (para reconstruir el .db)

Exportar:
time sqlite3 sakila.db ".output ./sakila.sql" ".dump"

Para abrir la base de datos con la shell de sqlite:
sqlite3 sakila.db

Para medir el tiempo (dentro de la shell de sqlite):
.timer ON

query 1:

```
SELECT description FROM film WHERE title="IRON MOON";
```

query 2:

```
SELECT COUNT(*) FROM film WHERE rating='R';
```

query 3:

```
SELECT AVG(amounts.actors_amount) AS actors_amount_avg
FROM (SELECT COUNT(a.actor_id) AS actors_amount
        FROM film f, film_actor a
        WHERE f.film_id=a.film_id AND f.length>=90 AND f.length<=120
        GROUP BY f.film_id) AS amounts;
```

query 4:

```
SELECT first_name, last_name, COUNT(rental_id) as rentals_amount
FROM customer c, rental r
WHERE c.customer_id=r.customer_id
GROUP BY c.customer_id
ORDER BY rentals_amount DESC LIMIT 10;
```

ventajas:
Muy fácil mover la base de datos así como copiarla, ya que toda esta contenida en un único archivo.

desventajas:
Más lento que MySQL, menos control sobre los datos (pocos tipos de datos), y más importante no permite conectarse a la base de datos imposibilitando su uso en un servidor externo.