DevOps L1 course

Database Administration

Lecture 2

Database Administration

Serge Prykhodchenko

QA: DevOps for Databases

- https://www.xenonstack.com/blog/devops-for-databases - взагалі

- https://www.devopsschool.com/blog/mysql-commands-cheatsheet-and-reference/ - збірка скриптів та завдань

- https://learn.microsoft.com/sr-cyrl-rs/azure/devops/pipelines/tasks/deploy/mysqldb-deployment?view=azure-devops&viewFallbackFrom=tfs-2013 – Azure devops course

Agenda

- DB Administration
- DB in clouds
- NonSQL-databases
- Q&A

DB ADMINISTRATION

History

Classical approaches to filling the concept of "DBA" began to form after the publication of the working report of the group on databases of the American National Standards Institute ANSI/X3/SPARC in 1975. This report described a three-tier DBMS architecture. This architecture defined three DBA roles: conceptual schema administrator, external schema administrator, and storage administrator. In a very small system, these roles could be played by one person; in a large system, a group of people could be assigned to each role. Each role was assigned a set of functions, and all these functions together constituted the functions of the DBA.

In 1980 - 1981 it became accepted in the American literature to include in the functions of the ADB:

- organizational and technical planning of the database,
- database design,
- providing support for application development,
- DB operation management.

DBA Administration function

- analysis of the subject area;
- database structure design, data integrity assurance;
- initial loading and maintenance of the database;
- data protection, user policies;
- work with users;
- analysis of user requests to the database;
- analysis of the effectiveness of the functioning of the SDS and the development and optimization of the system;
- ensuring the transition to a new version of the DBMS;
- database backup and recovery;
- organizational and methodological work.

Malfunctioning DBMS

One of the main requirements for a DBMS is reliable storage of data in external memory. Storage reliability refers to the fact that the DBMS must be able to restore the last consistent state of the DB after any hardware or software failure. Two possible types of hardware failures are commonly considered: so-called soft failures, which can be interpreted as a sudden shutdown of the computer (for example, an emergency power off), and hard failures, characterized by the loss of information on external memory media.

Examples of software failures can be a DBMS crash (due to an error in the program or some hardware failure) or a user program crash, as a result of which some transaction remains incomplete. The first situation can be viewed as a special kind of soft hardware failure; when the latter occurs, it is required to eliminate the consequences of only one transaction.

Journaling

To restore the database, you need to have some additional information. In other words, maintaining reliable data storage in a database requires redundant data storage, and that part of them that is used for recovery must be stored especially reliably. The most common method for maintaining such redundant information is to maintain a database change log.

The journal is a special part of the database that is inaccessible to DBMS users and is maintained very carefully (sometimes two copies of the journal are maintained, located on different physical disks), which receives records of all changes to the main part of the database. In different DBMSs, database changes are logged at different levels: sometimes a log entry corresponds to some logical operation of a database change (for example, an operation to delete a row from a relational database table), and sometimes a record corresponds to a minimal internal operation of modifying an external memory page. Some systems use both approaches simultaneously.

Write Ahead Log

In all cases, the strategy of "ahead of time" writing to the log (the so-called Write Ahead Log - WAL protocol) is followed. Roughly speaking, this strategy consists in the fact that a record about a change of any database object must get into the external memory of the log before the changed object gets into the external memory of the main part of the database. It is known that if the WAL protocol is correctly observed in the DBMS, then using the log you can solve all the problems of restoring the database after any failure.

The simplest recovery situation is an individual rollback of a transaction. Strictly speaking, this does not require a system-wide database changelog. It is enough for each transaction to maintain a local log of the database modification operations performed in this transaction, and to roll back the transaction by performing the reverse operations, following from the end of the local log. In some DBMSs they do this, but in most systems local logs do not support, and individual transactions are rolled back according to the system-wide log, for which all records from one transaction are linked in a reverse list (from end to beginning).

Soft failures

In the event of a soft failure, the external memory of the main part of the database may contain objects modified by transactions that were not completed at the time of the failure, and there may be no objects modified by transactions that had successfully completed by the time of the failure (due to the use of RAM buffers, the contents of which disappear during a soft failure). If you follow the WAL protocol, the external log memory must be guaranteed to contain records related to the modification operations of both types of objects. The goal of the recovery process after a soft failure is the state of the external memory of the main part of the database, which would arise when all completed transactions were committed to external memory, and which would not contain any traces of unfinished transactions. To achieve this, they first roll back uncommitted transactions (undo), and then replay (redo) those operations of completed transactions whose results are not mapped to external memory. This process contains many subtleties related to the overall organization of buffer and log management.

Hard failures

To restore the database after a hard failure, a log and an archive copy of the database are used. Roughly speaking, an archive copy is a complete copy of the database by the time the journal starts filling (there are many options for a more flexible interpretation of the meaning of an archive copy). Of course, for a normal database recovery after a hard failure, it is necessary that the log does not disappear. As already noted, especially increased requirements are imposed on the safety of the journal in external memory in the DBMS. Then database recovery consists in the fact that, based on the archive copy, the work of all transactions that had ended by the time of the failure is reproduced in the log. In principle, you can even reproduce the work of incomplete transactions and continue their work after the end of recovery. However, in real systems this is usually not done because the recovery process from a hard failure is quite long.

Access control

- After obtaining the right to access the DBMS, the user automatically receives the **privileges** associated with his identifier. This can relate to procedures for accessing database objects, to operations on data. For the main objects of the database, tables can be built, which indicate the set of actions available to each user of the system.

- Each possible action on the data in the table is assigned a binary value, the overall result of possible operations is obtained by summing the values entered by the user.

- Privileges in a DBMS can be divided into two categories: **security privileges** and **access privileges**. Security privileges allow you to perform administrative actions, access privileges determine the access rights of subjects to certain objects.

- Before proceeding with the assignment of privileges, they must be created.

Privilege

Privileges can be subdivided according to the **types of objects** to which they belong: tables and views, procedures, databases, database server.

With regard to **tables**, the following access rights can be defined: the right to select, delete, update, add, the right to use foreign keys that refer to this table. By default, the user does not have any access rights to tables or views.

For **procedures**, you can grant the right to execute them, but you do not specify privileges on the right to access objects processed by the procedures. This allows you to allocate uncontrolled access to perform well-defined operations on the data.

In relation to the **database**, the allocated rights are actually prohibitive: a limit on the number of row I / O operations, the number of rows returned by one query.

Logging and auditing

Audit - checking that all provided controls are in place and meet the level of security specified.

Such a measure as **logging and auditing** consists in the following: detection of unusual and suspicious user actions and identification of the persons who committed these actions; assessment of the possible consequences of the violation; Giving help; organization of passive protection of information from illegal user actions: maintaining the accuracy of the entered data; active documentation support; correct user testing.

It is recommended that when organizing logging, record the facts of transfer of privileges and connections to a particular database.

MySQL Server Administration

- Server configuration
- The data directory, particularly the mysql system schema
- The server log files
- Management of multiple servers on a single machine

https://dev.mysql.com/doc/refman/8.0/en/server-administration.html

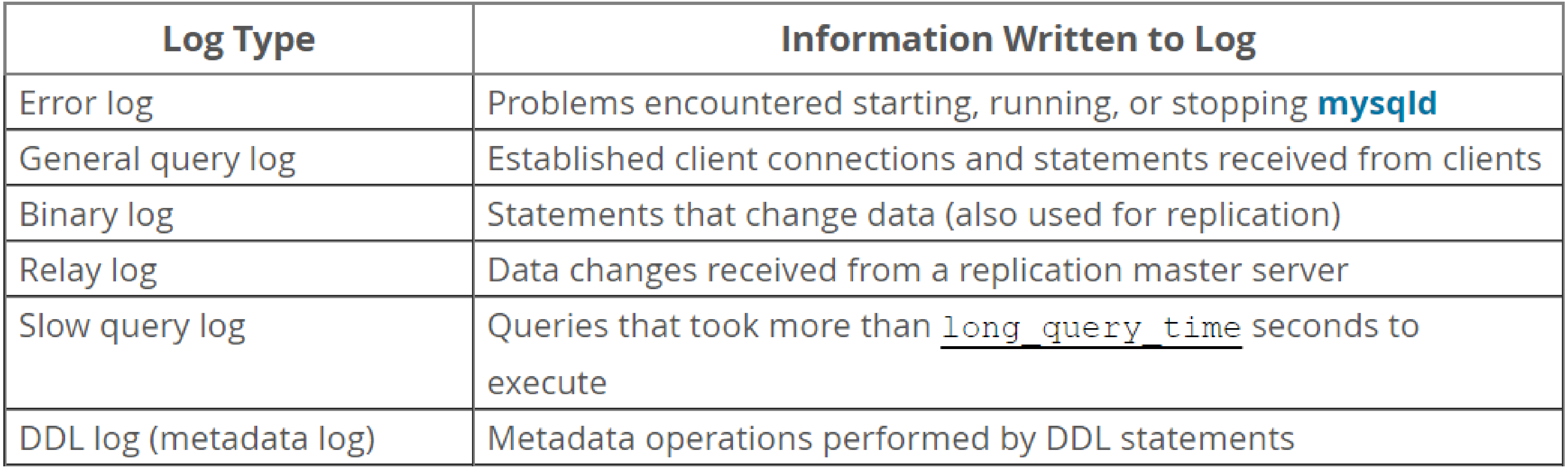
MySQL Database Server Administration (Job Example)

- Optimizing MySQL parameters for maximum performance.
- Updating the operating system and installed software.
- Automated monitoring of the availability of the server and the services it provides, and reporting of failures. Working mode 24 x 7.
- Automated monitoring of the server for software and hardware failures, and reporting failures.

Working mode 24 x 7.

- Organization of backup of main files for subsequent quick recovery of the server in case of failure.
- Securing the server from unauthorized access (setting iptables, fail2ban).
- Diagnostics of the reasons for slow work, slowlog analysis.
- Prompt response in case of server inoperability within 30 minutes.

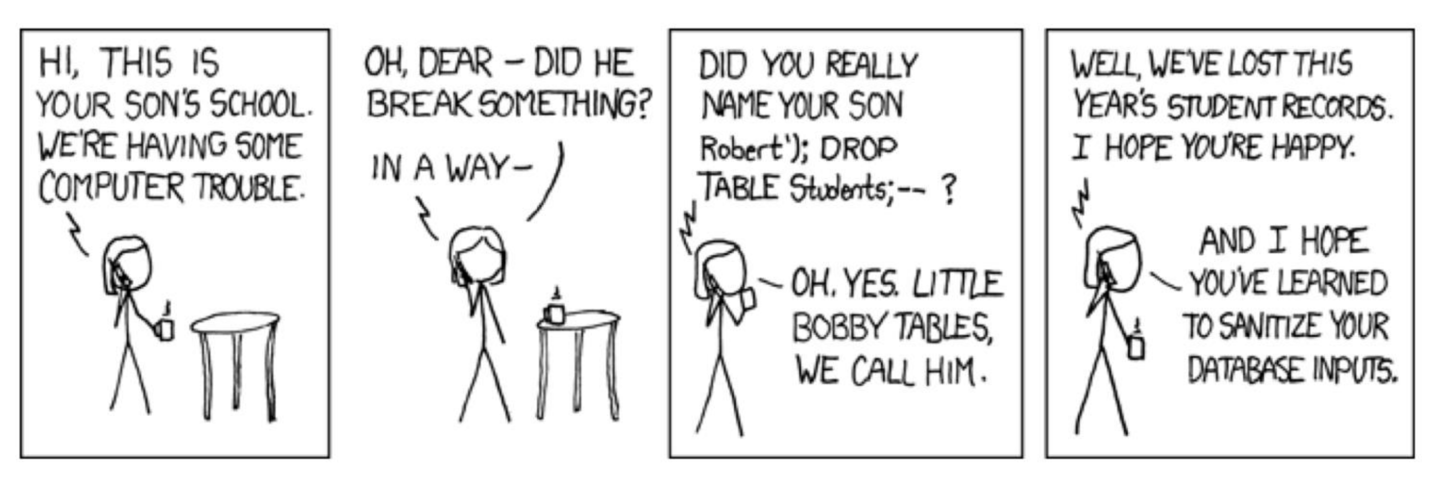
Logs



Log Type	Information Written to Log
Error log	Problems encountered starting, running, or stopping **mysqld**
General query log	Established client connections and statements received from clients
Binary log	Statements that change data (also used for replication)
Relay log	Data changes received from a replication master server
Slow query log	Queries that took more than `long_query_time` seconds to execute
DDL log (metadata log)	Metadata operations performed by DDL statements

Security in DB Applications

- The most common form of security violation on the web involves an SQL injection attack on a DB-driven web site


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Security in DB Applications

- An SQL Injection attack *only* works when data provided by users (such as the contents of a form) is inserted directly into SQL and submitted to the database system
- To avoid this, any input should be cleaned by
  - Removing any SQL reserved characters (like “‘”, “;”, “)”, etc.
  - And possibly reserved words like “SELECT”, “DROP”, “TABLES”, etc.

Security in DB Applications

- Some Web Application Servers, like PHP, include functions to “sanitize” inputs
  - For example
    - mysql_real_escape_string($cname)
    - This basically just escapes quotes
    - Input string 'Louis'; DROP TABLE NEWCUST;'
    - Converted query ‘SELECT * FROM DIVECUST D where D.Name like '%Louis\'; DROP TABLE NEWCUST;%' ;

Monitoring

```
SHOW QUERY LOG
SHOW PROCCESSLIST
SHOW VARIABLES
SHOW GLOBAL STATUS
SHOW GLOBAL STATUS LIKE "Questions";
show global variables like 'slow%log%';
```

https://www.datadoghq.com/blog/monitoring-mysql-performance-metrics/

https://www.metricfire.com/blog/a-modern-guide-to-mysql-performance-monitoring/

Mysql system tables

The mysql database, which is used for server administration, contains 24 system tables (tables of privileges, performance, etc.)

Show databases;
Use mysql;
show tables;

User table
Determines whether the user trying to connect to the server is allowed to do this. Contains username, password, and privileges.
show columns from user;

User table

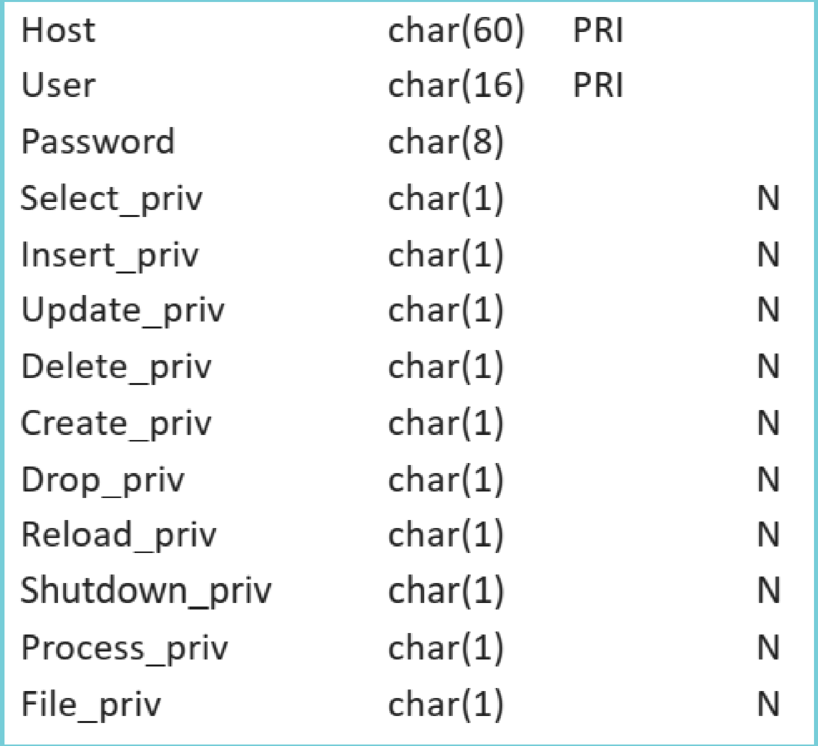
User table

Determines whether the user trying to connect to the server is allowed to do this. Contains username, password, and privileges.

show columns from user;

Initially this table contains the root user with the password you set and the hostname '%'. By default, root can log in from any host and has full privileges and access to all databases. The table also contains an entry for the user '%', which must be deleted immediately, since it provides access to any user

delete from user where user = '%';



Host	char(60)	PRI	
User	char(16)	PRI	
Password	char(8)		
Select_priv	char(1)		N
Insert_priv	char(1)		N
Update_priv	char(1)		N
Delete_priv	char(1)		N
Create_priv	char(1)		N
Drop_priv	char(1)		N
Reload_priv	char(1)		N
Shutdown_priv	char(1)		N
Process_priv	char(1)		N
File_priv	char(1)		N

db table

- Determines which databases which users and from which hosts are allowed to access. In this table, you can grant each user access to databases and assign privileges.

```
show columns from db;
```

- By default, all privileges are set to 'N'. For example, let's give the user john access to the library database and give him select, insert and update privileges.

```
insert into db (host, user, db, select_priv, insert_priv, update_priv) values
                                        ('% .domain.com', 'john', 'library', 'Y', 'Y', 'Y’);
```

- The privileges set on the db table only apply to the library database. If you set these privileges in the user table, then they will be distributed to other databases, even if access to them is not explicitly set.

mysql users

CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'password’;

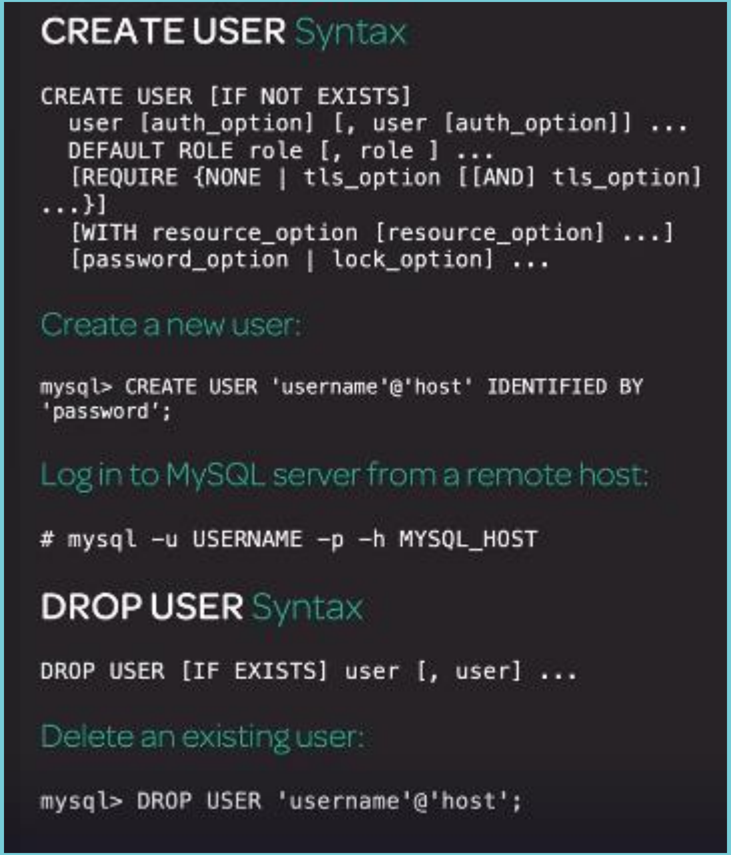
GRANT ALL PRIVILEGES ON * . * TO 'newuser'@'localhost’;

SHOW GRANTS FOR username;

REVOKE permission_type ON database.table TO 'username'@'localhost';

FLUSH PRIVILEGES;

Drop user 'newuser'@'localhost’ ;



CREATE USER Syntax

```
CREATE USER [IF NOT EXISTS]
  user [auth_option] [, user [auth_option]] ...
  DEFAULT ROLE role [, role ] ...
  [REQUIRE {NONE | tls_option [[AND] tls_option]
...}]
  [WITH resource_option [resource_option] ...]
  [password_option | lock_option] ...
```

Create a new user:

```
mysql> CREATE USER 'username'@'host' IDENTIFIED BY
'password';
```

Log in to MySQL server from a remote host:

```
# mysql -u USERNAME -p -h MYSQL_HOST
```

DROP USER Syntax

```
DROP USER [IF EXISTS] user [, user] ...
```

Delete an existing user:

```
mysql> DROP USER 'username'@'host';
```

mysql user permissons

Permissions are actions that the user is allowed to perform in the database. Depending on how much authority you want your user to have, you can grant them one, several or all of the following privileges:

- **All Privileges:** The user account has full access to the database
- **Insert:** The user can insert rows into tables
- **Delete:** The user can remove rows from tables
- **Create:** The user can create entirely new tables and databases
- **Drop:** The user can drop (remove) entire tables and databases
- **Select:** The user gets access to the select command, to read the information in the databases
- **Update:** The user can update table rows
- **Grant Option:** The user can modify other user account privileges

The basic syntax used to grant privileges to a user account is:

```
GRANT permission_type ON database.table TO 'username'@'localhost';
```

mysql transactions

A transaction in MySQL is a **sequential group of statements**, queries, or operations such as select, insert, update or delete to perform as a one single work unit that can be committed or rolled back. If the transaction makes multiple modifications into the database, two things happen:

- Either all modification is successful when the transaction is committed.
- Or, all modifications are undone when the transaction is rollback.

In other words, a transaction cannot be successful without completing each operation available in the set. It means if any statement fails, the transaction operation cannot produce results.

mysql transactions

```
-- 1. Start a new transaction
START TRANSACTION;
-- 2. Get the highest income
SELECT @income:= MAX(income) FROM employees;
-- 3. Insert a new record into the employee table
INSERT INTO employees(emp_id, emp_name, emp_age, city, income)
VALUES (111, 'Alexander', 45, 'California', 70000);
-- 4. Insert a new record into the order table
INSERT INTO Orders(order_id, prod_name, order_num, order_date)
VALUES (6, 'Printer', 5654, '2020-01-10');
-- 5. Commit changes
COMMIT;
```

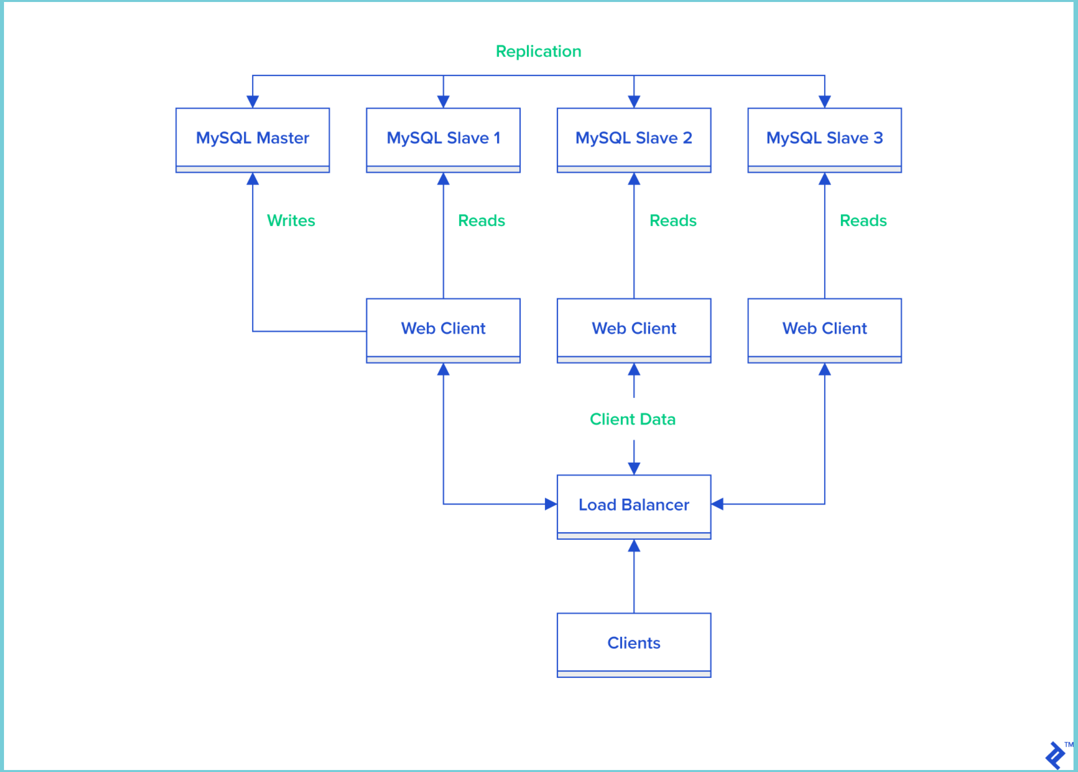
mysql replication

MySQL replication is a process that enables data from one MySQL database server (the master) to be copied automatically to one or more MySQL database servers (the slaves). It is usually used to spread read access on multiple servers for scalability, although it can also be used for other purposes such as for failover, or analyzing data on the slave in order not to overload the master.

As the master-slave replication is a one-way replication (from master to slave), only the master database is used for the write operations, while read operations may be spread on multiple slave databases. What this means is that if master-slave replication is used as the scale-out solution, you need to have at least two data sources defined, one for write operations and the second for read operations.

mysql replication

https://www.toptal.com/mysql/mysql-master-slave-replication-tutorial



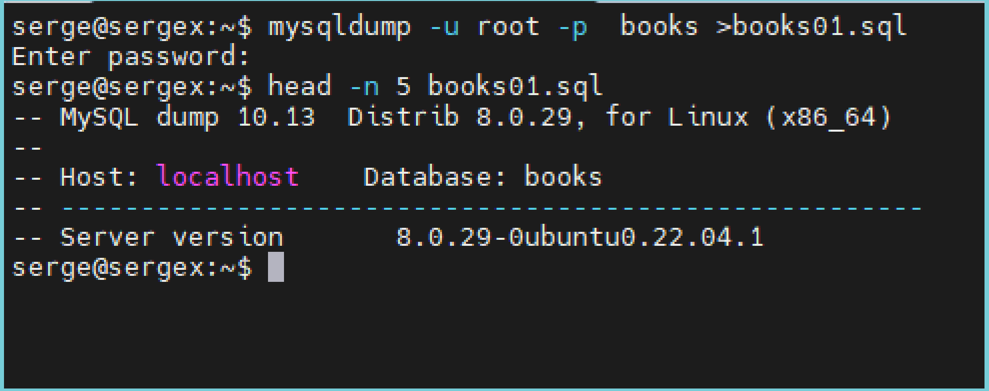
Backup and Recovery Types

- Physical (Raw) Versus Logical Backups
- Online Versus Offline Backups
- “hot” versus “cold” versus “warm”
- Local Versus Remote Backups
- Snapshot Backups (for MySQL is available through third-party solutions such as Veritas, LVM, or ZFS)
- Full Versus Incremental Backups
- Full Versus Point-in-Time (Incremental) Recovery
- Backup Scheduling, Compression, and Encryption

https://dev.mysql.com/doc/refman/8.0/en/backup-types.html mysqldump

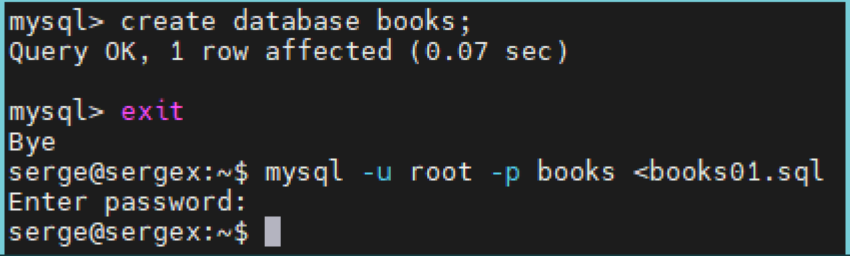
mysql backup

$ mysqldump -u user -p dbname >bkup.sql



```
serge@sergex:~$ mysqldump -u root -p  books >books01.sql
Enter password:
serge@sergex:~$ head -n 5 books01.sql
-- MySQL dump 10.13  Distrib 8.0.29, for Linux (x86_64)
--
-- Host: localhost    Database: books
-- ------------------------------------------------------
-- Server version       8.0.29-0ubuntu0.22.04.1
serge@sergex:~$
```

$ mysql -u user -p dbname <bkup.sql



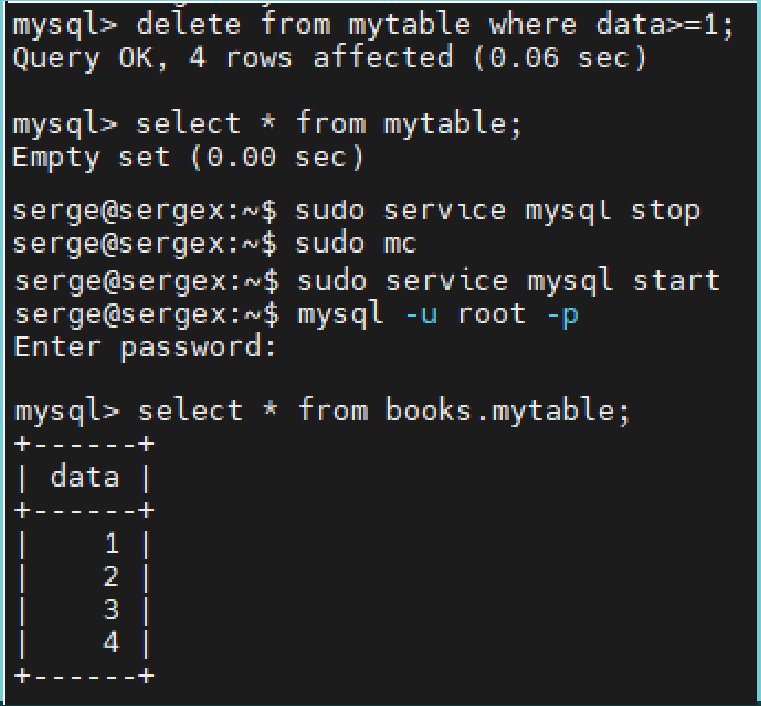
```
mysql> create database books;
Query OK, 1 row affected (0.07 sec)

mysql> exit
Bye
serge@sergex:~$ mysql -u root -p books <books01.sql
Enter password:
serge@sergex:~$
```

mysql backup

```
mysqldump --databases db_1 db_2 > db_backup.sql
mysqldump --all-databases > db_backup.sql
```

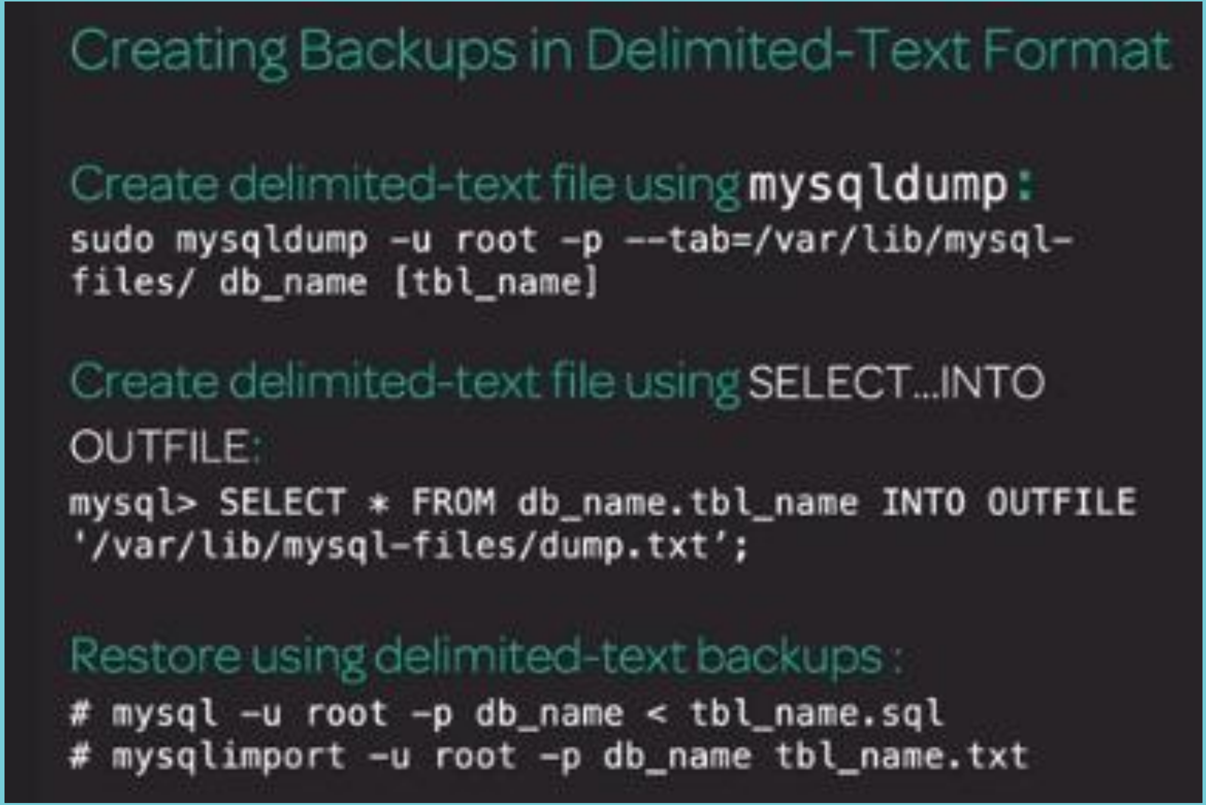
```
mysqlshow -uroot -p
```

```
cp /var/lib/mysql/db/* …
```



```
mysql> delete from mytable where data>=1;
Query OK, 4 rows affected (0.06 sec)

mysql> select * from mytable;
Empty set (0.00 sec)

serge@sergex:~$ sudo service mysql stop
serge@sergex:~$ sudo mc
serge@sergex:~$ sudo service mysql start
serge@sergex:~$ mysql -u root -p
Enter password:

mysql> select * from books.mytable;
+------+
| data |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
+------+
```

mysql backup



Creating Backups in Delimited-Text Format

Create delimited-text file using **mysqldump**:

```
sudo mysqldump -u root -p --tab=/var/lib/mysql-
files/ db_name [tbl_name]
```

Create delimited-text file using SELECT...INTO OUTFILE:

```
mysql> SELECT * FROM db_name.tbl_name INTO OUTFILE
'/var/lib/mysql-files/dump.txt';
```

Restore using delimited-text backups :

```
# mysql -u root -p db_name < tbl_name.sql
# mysqlimport -u root -p db_name tbl_name.txt
```

Executing SQL Statements from a Text File

The mysql client typically is used interactively, like this:

```
mysql db_name
```

However, it is also possible to put your SQL statements in a file and then tell mysql to read its input from that file. To do so, create a text file text_file that contains the statements you wish to execute. Then invoke mysql as shown here:

```
mysql db_name < text_file
```

If you place a **USE db_name** statement as the first statement in the file, it is unnecessary to specify the database name on the command line:

```
mysql < text_file
```

If you are already running mysql, you can execute an SQL script file using the source command or \. command:

```
mysql> source file_name
mysql> \. file_name
```

https://dev.mysql.com/doc/refman/8.0/en/mysql-batch-commands.html

AWS DATABASE TYPES

Management

- **Unmanaged** – managed by you

example: set up EC2, install DB into EC2

+ you have more fine-tuned control over how your solution handles changes in load, errors, and situations where resources become unavailable

- **Managed** - Scaling, fault tolerance, and availability are typically built into the service.

example: set up RDS

+ You manage: Application optimization

AWS manages: OS installation and patches; Database software installation and patches; Database backups; High availability; Scaling; Power and racking and stacking servers; Server maintenance

Database types

Database type	Use cases	AWS service
Relational	Traditional applications, ERP, CRM, e-commerce	Amazon Aurora Amazon RDS Amazon Redshift
Key-value	High-traffic web apps, e-commerce systems, gaming applications	Amazon DynamoDB
In-memory	Caching, session management, gaming leaderboards, geospatial applications	Amazon ElastiCache for Memcached /for Redis
Document	Content management, catalogs, user profiles	Amazon DocumentDB
Graph	Fraud detection, social networking, recommendation engines	Amazon Neptune
Time Series	IoT applications, DevOps, industrial telemetry	Amazon Timestream
Ledger	Systems of record, supply chain, registrations, banking transactions	Amazon QLDB

Application requires

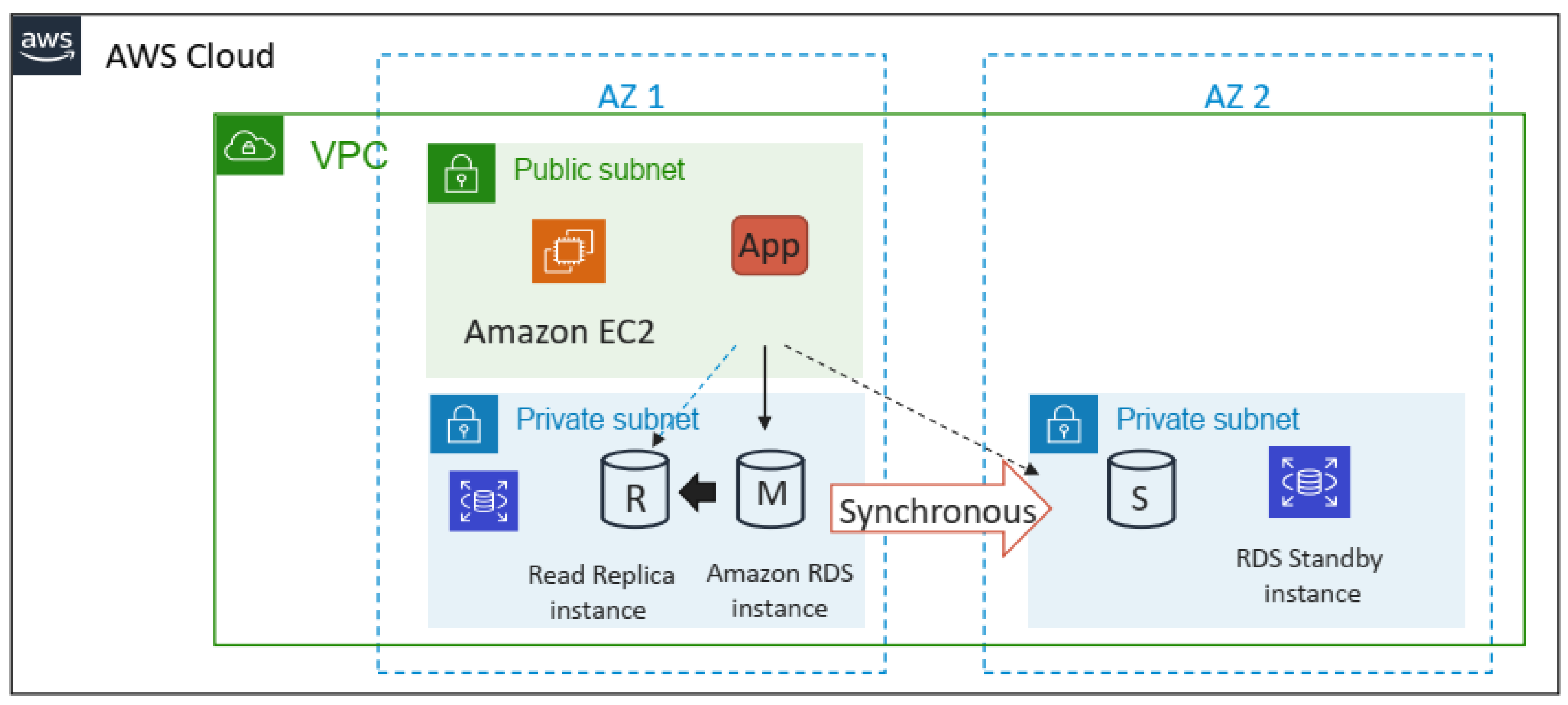
- Complex transactions or complex queries
- A medium to high query or write rate – Up to 30,000 IOPS (15,000 reads + 15,000 writes)
- No more than a single worker node or shard
- High durability

☺ Use Amazon RDS

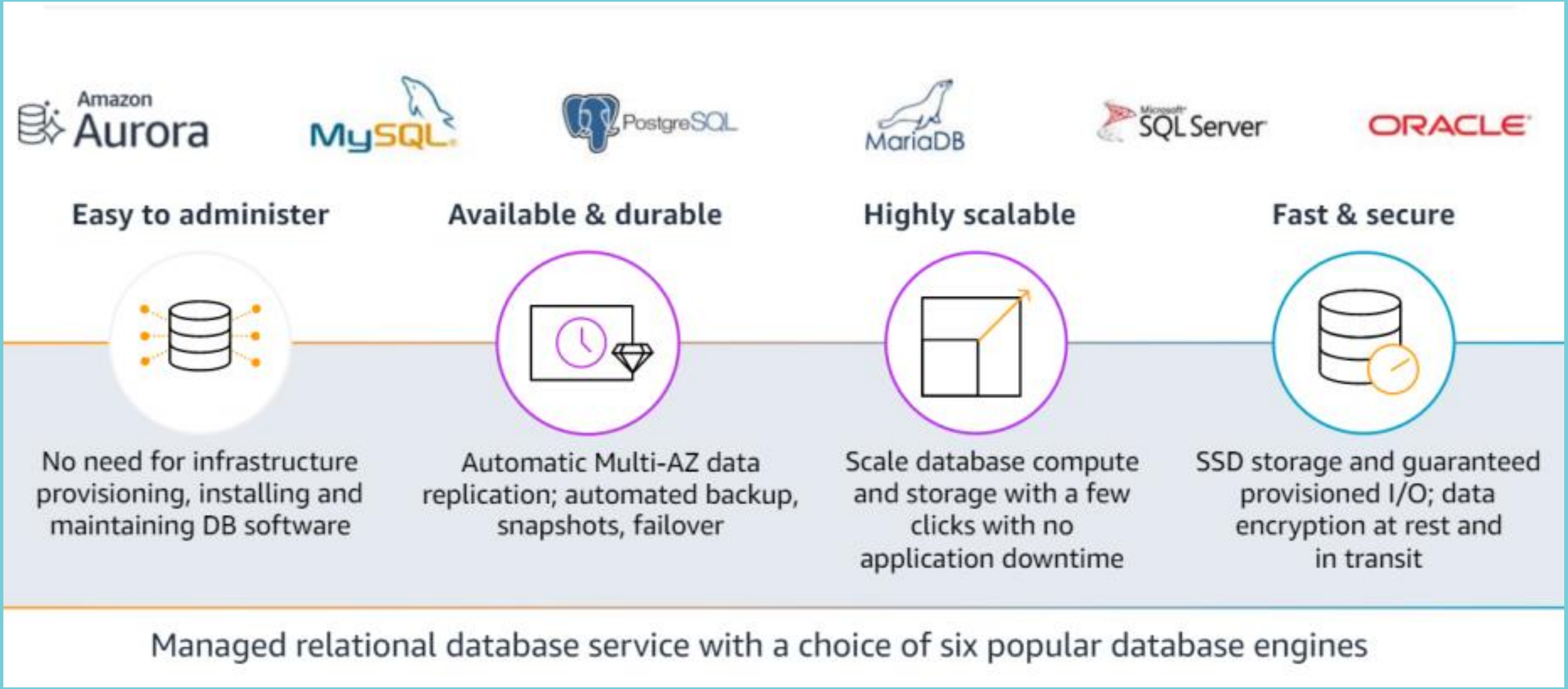
- Massive read/write rates (for example, 150,000 write/second)
- Sharding due to high data size or throughput demands
- Simple GET or PUT requests and queries that a NoSQL database can handle
- Relational database management system (RDBMS) customization

☺ Do not Use Amazon RDS

Amazon RDS in VPC



Amazon RDS



Easy to administer

No need for infrastructure provisioning, installing and maintaining DB software

Available & durable

Automatic Multi-AZ data replication; automated backup, snapshots, failover

Highly scalable

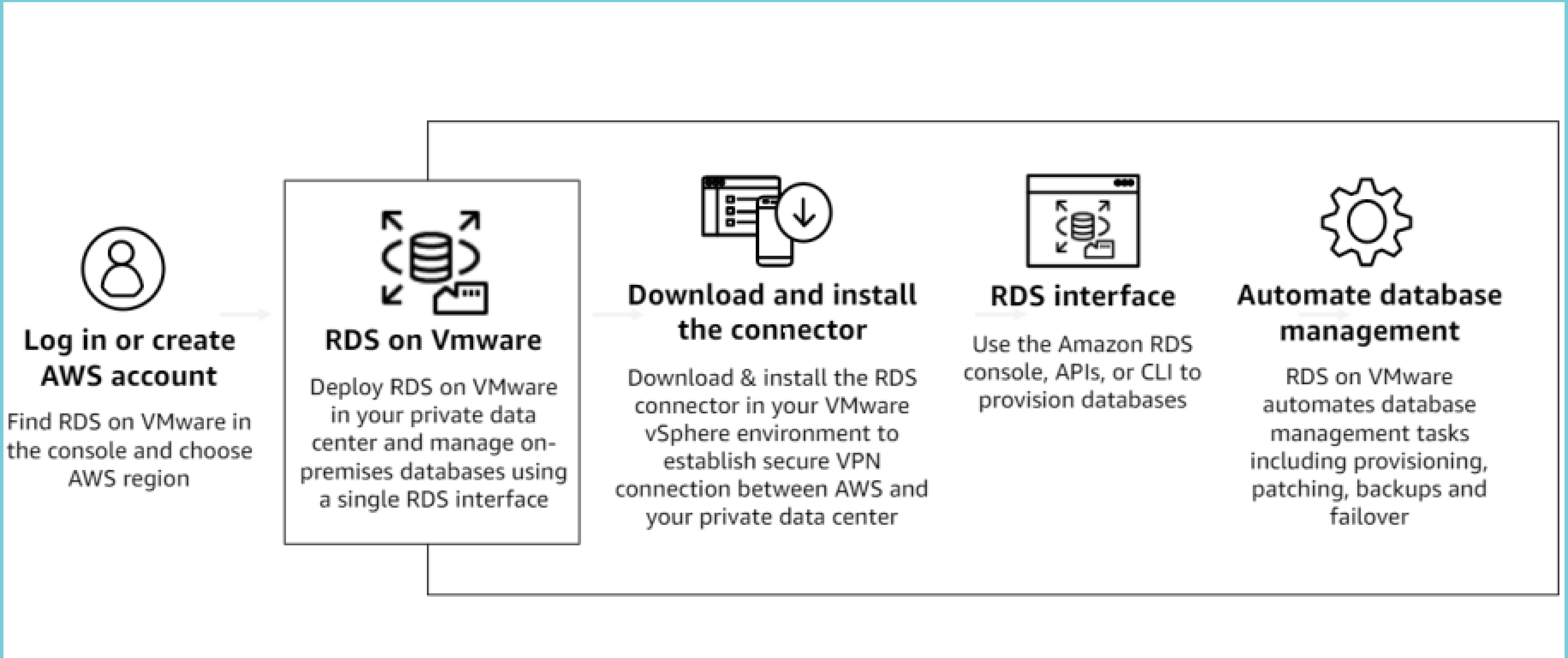
Scale database compute and storage with a few clicks with no application downtime

Fast & secure

SSD storage and guaranteed provisioned I/O; data encryption at rest and in transit

Managed relational database service with a choice of six popular database engines

How it work



Log in or create AWS account

Find RDS on VMware in the console and choose AWS region

RDS on Vmware

Deploy RDS on VMware in your private data center and manage on-premises databases using a single RDS interface

Download and install the connector

Download & install the RDS connector in your VMware vSphere environment to establish secure VPN connection between AWS and your private data center

RDS interface

Use the Amazon RDS console, APIs, or CLI to provision databases

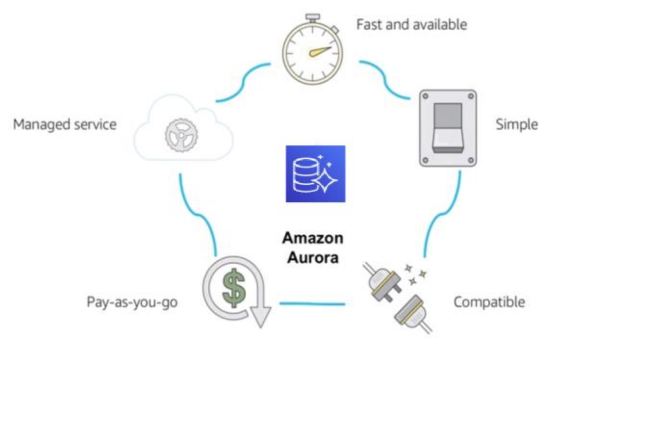
Automate database management

RDS on VMware automates database management tasks including provisioning, patching, backups and failover

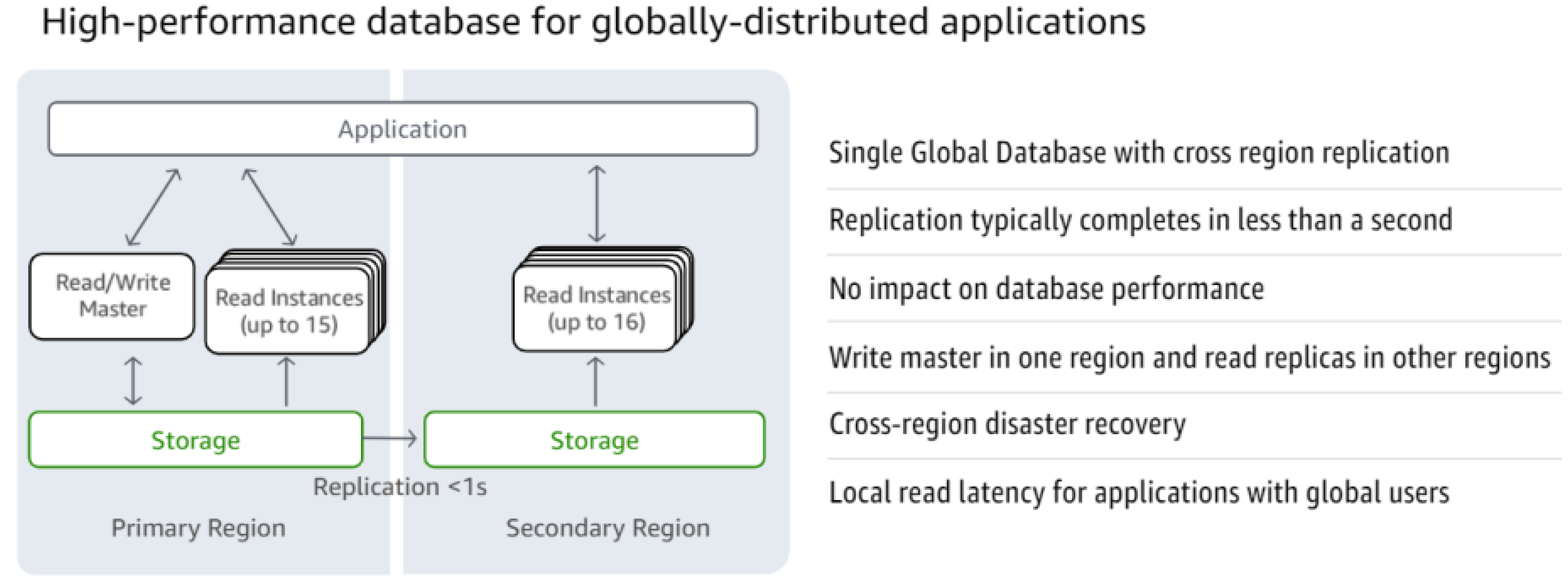
Amazon Aurora

- Enterprise-class relational database
- Compatible with MySQL or PostgreSQL
- Automate time-consuming tasks (such as provisioning, patching, backup, recovery, failure detection, and repair).

Amazon Aurora



Amazon Aurora



High-performance database for globally-distributed applications

Single Global Database with cross region replication

Replication typically completes in less than a second

No impact on database performance

Write master in one region and read replicas in other regions

Cross-region disaster recovery

Local read latency for applications with global users

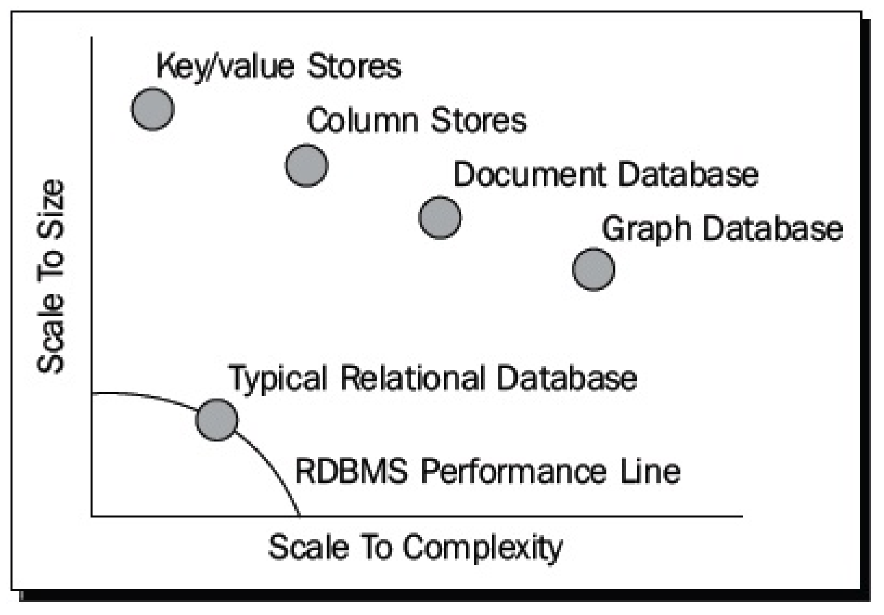
NONSQL DATABASES

Use cases

- High performance and scalable applications
- Most web applications where you would previously use SQL

Do not use for:

- Transaction-critical applications



Document databases

Advantages

Flexible, semi-structured, and hierarchical

Adjustable to application needs as databases evolve

Flexible schema

Simple hierarchical and semi-structured data

Powerfully index for fast querying

Naturally map documents to object-oriented programming

Easily flows data to persistent layer

Expressive query languages built for documents

Capable of ad-hoc queries and aggregations across documents

Use Cases

Catalogs

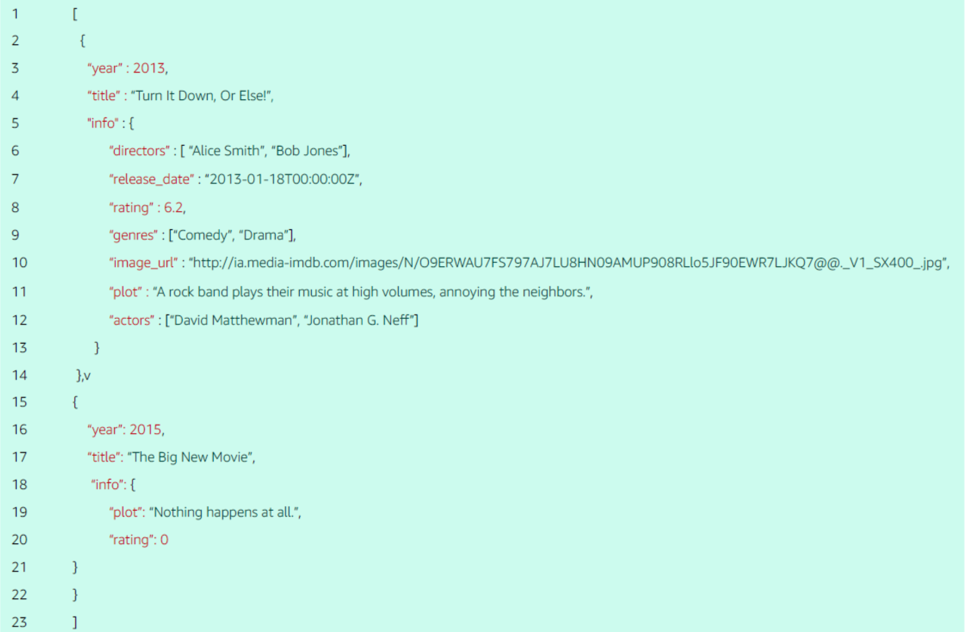
Content management systems

User profiles/personalization

Mobile

Not designed for

Explicitly defined relations between different pieces of data



```
[
 {
  "year" : 2013,
  "title" : "Turn It Down, Or Else!",
  "info" : {
      "directors" : [ "Alice Smith", "Bob Jones"],
      "release_date" : "2013-01-18T00:00:00Z",
      "rating" : 6.2,
      "genres" : ["Comedy", "Drama"],
      "image_url" : "http://ia.media-imdb.com/images/N/O9ERWAU7FS797AJ7LU8HN09AMUP908RLlo5JF90EWR7LJKQ7@@._V1_SX400_.jpg",
      "plot" : "A rock band plays their music at high volumes, annoying the neighbors.",
      "actors" : ["David Matthewman", "Jonathan G. Neff"]
    }
 },v
{
  "year": 2015,
  "title": "The Big New Movie",
   "info": {
      "plot": "Nothing happens at all.",
      "rating": 0
}
}
]
```

MongoDB

- Database for JSON objects
  - Perfect as a simple persistence layer for JavaScript objects
  - “NoSQL database”
- Data is stored as a collection of documents
  - Document: (almost) JSON object
  - Collection: group of “similar” documents
- Analogy
  - Document in MongoDB ~ row in RDB
  - Collection in MongoDB ~ table in RDB

MongoDB “Document”

```
{
    "_id": ObjectId(8df38ad8902c),
    "title": "MongoDB",
    "description": "MongoDB is NoSQL database",
    "tags": ["mongodb", "database", "NoSQL"],
    "likes": 100,
    "comments": [
        { "user":"lover", "comment": "Perfect!" },
        { "user":"hater", "comment": "Worst!" }
    ]
}
```

- _id field: primary key
  - May be of any type other than array
  - If not provided, automatically added with a unique ObjectId value
- Stored as BSON (Binary representation of JSON)
  - Supports more data types than JSON
  - Does not require double quotes for field names

MongoDB “Philosophy”

- Adopts JavaScript “laissez faire” philosophy
  - Don’t be too strict! Be accommodating! Handle user request in a “reasonable” way
- Schema-less: no predefined schema
  - Give me anything. I will store it anywhere you want
  - One collection will store documents of *any* kind with no complaint
- No need to “plan ahead”
  - A “database” is created when a first collection is created
  - A “collection” is created when a first document is inserted
- Both blessing and curse

MongoDB Demo

```
show dbs;
use demo;
show collections;
db.books.insertOne({title: "MongoDB", likes: 100});
db.books.find();
show collections;
show dbs;
db.books.insertMany([{title: "a"}, {name: "b"}]);
db.books.find();
db.books.find({likes: 100});
db.books.find({likes: {$gt: 10}});
db.books.updateOne({title: "MongoDB"}, {$set: { likes: 200 }});
db.books.find();
db.books.deleteOne({title: "a"});
db.books.drop();
show collections;
show dbs;
```

Querying

- Queries return a cursor, which can be iterated to retrieve results
- Query optimizer executes new plans in parallel
- Queries are expressed as BSON documents which indicate a query pattern

```
db.users.find({'last_name': 'Smith'})

// retrieve ssn field for documents where last_name == 'Smith':
db.users.find({last_name: 'Smith'}, {'ssn': 1});

// retrieve all fields *except* the thumbnail field, for all documents:
db.users.find({}, {thumbnail:0});

// retrieve all users order by last_name:
db.users.find({}).sort({last_name: 1});

// skip and limit:
db.users.find().skip(20).limit(10);
```

Advanced querying

```
{ name: "Joe", address: { city: "San Francisco", state: "CA"
} , likes: [ 'scuba', 'math', 'literature' ] }

// field in sub-document:
db.persons.find( { "address.state" : "CA" } )

// find in array:
db.persons.find( { likes : "math" } )

// regular expressions:
db.persons.find( { name : /acme.*corp/i } );

// javascript where clause:
db.persons.find("this.name != 'Joe'");

// check for existence of field:
db.persons.find( { address : { $exists : true } } );
```

- Aggregate queries like group by, count, distinct; only available for single instances

In-memory databases

Advantages

Sub-millisecond latency

Can perform millions of operations per second

Significant performance gains when compared to disk-based alternatives

Simpler instruction set

Support for rich command set (Redis)

Works with any type of database, relational or non-relational, or even storage services

Use Cases

Caching

Session store

Gaming

Leaderboards

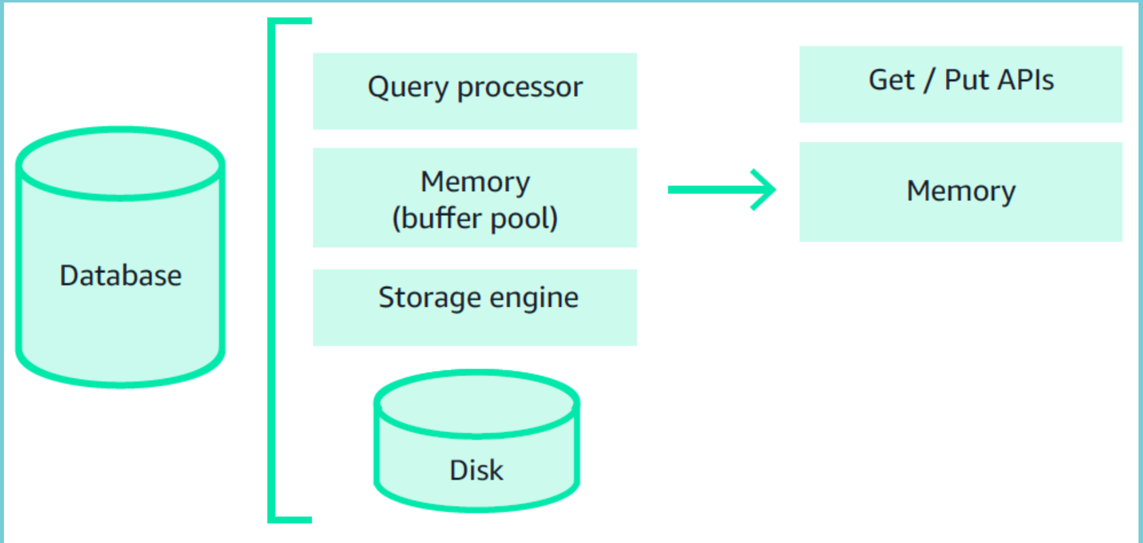
Geospatial services

Pub/sub

Real-time streaming

Not designed for

Persisting data to disk all the time



Graph databases

Advantages

Ability to make frequent schema changes

Quickly make relationships between many different types of data

Real-time query response time

Superior performance for querying related data—big or small

Meets more intelligent data activation requirements

Explicit semantics for each query—no hidden assumptions

Flexible online schema environment

Use Cases

Fraud detection

Social networking

Recommendation engines

Knowledge graphs

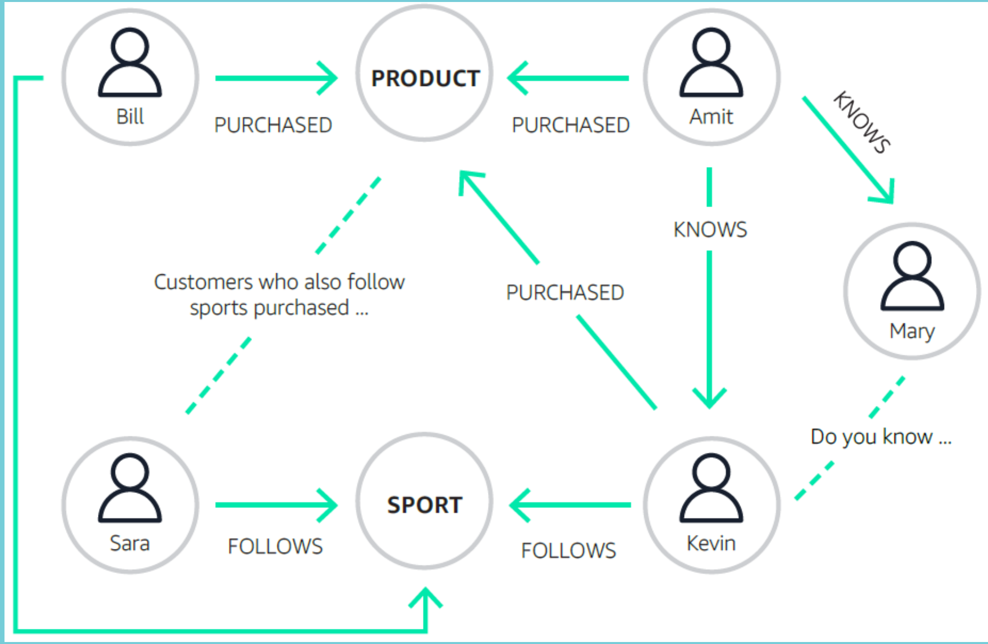
Data lineage

Not designed for

Applications that do not traverse or query relationships

Processing high volumes of transactions

Handling queries that span the entire database



Time-series databases

Advantages

Ideal for measurements or events that are tracked, monitored, and aggregated over time

High scalability for quickly accumulating time-series data

Robust usability for many functions, including: data-retention policies, continuous queries, flexible-time aggregations

Use Cases

DevOps

Application monitoring

Industrial telemetry

IoT applications

Not designed for

Data not in time-order form, such as: documents, catalogs, customer profiles

Ledger databases

Advantages
Maintain accurate history of application data
Immutable and transparent
Cryptographically verifiable
Highly scalable

Use Cases
Finance - Keep track of ledger data such as credits and debits
Manufacturing - Reconcile data between supply chain systems to track full manufacturing history
Insurance - Accurately track claims history
HR and payroll - Track and maintain a record of employee details
Retail - Maintain an accurate log of inventory

Not designed for
Decentralized use case (i.e., multiple entities need to read/write on data independently)

QUESTIONS & ANSWERS

COMMON LINKS

DB intro (eng) - https://open.umn.edu/opentextbooks/textbooks/354
DB intro (ua) - https://web.posibnyky.vntu.edu.ua/fitki/11petuh_bazdanyh_movy_zalitiv/zmist.htm
DB intro (ru) - http://citforum.ru/database/advanced_intro/

NoSQL - https://www.analyticsvidhya.com/blog/2020/09/different-nosql-databases-every-data-scientist-must-know/

LINKS FOR TASK

3. https://www.educative.io/blog/what-are-database-schemas-examples
4. https://www.mysqltutorial.org/mysql-create-database/
5. https://www.sqlshack.com/learn-sql-insert-into-table/
6. https://dev.mysql.com/doc/refman/8.0/en/select.html
7. https://www.tutorialgateway.org/sql-dml-ddl-dcl-and-tcl-commands/
8. https://chartio.com/resources/tutorials/how-to-grant-all-privileges-on-a-database-in-mysql/
https://dev.mysql.com/doc/refman/8.0/en/grant.html
9. Look at p.6

10. https://support.hostway.com/hc/en-us/articles/360000220190-How-to-backup-and-restore-MySQL-databases-on-Linux
11. https://phoenixnap.com/kb/mysql-drop-table
12. Look at p.10

LINKS FOR TASK

13. https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_GettingStarted.CreatingConnecting.MySQL.html
https://docs.bitnami.com/aws/how-to/migrate-database-rds/
14. https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_GettingStarted.CreatingConnecting.MySQL.html
15. Look at p.6
16. Look at p.10
17-20. http://nicholasjohnson.com/mongo/course/workbook/

THANK YOU!