```
** PATH **

# temporarily add mysql binaries to shell path so mysql command works without
full path
export PATH=${PATH}:/usr/local/mysql/bin

# permanently add mysql binaries to path for all future terminal sessions
echo 'export PATH="/usr/local/mysql/bin:$PATH"' >> ~/.bash_profile



** SERVICE **

# start mysql database service so server begins accepting connections
systemctl start mysql

# stop mysql service completely (no connections possible)
systemctl stop mysql

# restart mysql service to apply config changes or fix issues
systemctl restart mysql

# check current status, errors, and whether mysql is running
systemctl status mysql



** LOGIN **

# login to mysql as root with password prompt
mysql -u root -p

# login directly into a specific database as a user
mysql -u user -p db_name

# exit mysql shell back to linux terminal
exit;

# display currently authenticated mysql user
SELECT USER();


** USERS **

# list all mysql users and allowed host connections
SELECT User, Host FROM mysql.user;

# create a new database user with password
CREATE USER 'someuser'@'localhost' IDENTIFIED BY 'somepassword';

# grant full access to all databases and tables for user
GRANT ALL PRIVILEGES ON *.* TO 'someuser'@'localhost';

# reload privilege tables so changes take effect immediately
FLUSH PRIVILEGES;

# show what permissions a user currently has
SHOW GRANTS FOR 'someuser'@'localhost';

# remove all privileges from a user (access revoked)
REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'someuser'@'localhost';

# delete user account completely from mysql
```

```
DROP USER 'someuser'@'localhost';


** DATABASES **

# display all databases available on server
SHOW DATABASES;

# create a new empty database
CREATE DATABASE acme;

# permanently delete a database and all its data
DROP DATABASE acme;

# switch active database for current session
USE acme;


** TABLES **

# create a table with columns and constraints to store user data
CREATE TABLE users(
id INT AUTO_INCREMENT,
first_name VARCHAR(100),
last_name VARCHAR(100),
email VARCHAR(50),
password VARCHAR(20),
location VARCHAR(100),
dept VARCHAR(100),
is_admin TINYINT(1),
register_date DATETIME,
PRIMARY KEY(id)
);

# delete a table and all stored records
DROP TABLE tablename;

# list all tables in current database
SHOW TABLES;

# show table structure including columns and types
DESCRIBE users;


** INSERT **

# insert a single new record into table
INSERT INTO users (first_name, last_name, email, password, location, dept,
is_admin, register_date) values ('Brad', 'Traversy', 'brad@gmail.com',
'123456','Massachusetts', 'development', 1, now());

# insert multiple records in one query for efficiency
INSERT INTO users (first_name, last_name, email, password, location, dept,
is_admin, register_date) values ('Fred', 'Smith', 'fred@gmail.com', '123456',
'New York', 'design', 0, now()), ('Sara', 'Watson', 'sara@gmail.com', '123456',
'New York', 'design', 0, now()),('Will', 'Jackson', 'will@yahoo.com', '123456',
'Rhode Island', 'development', 1, now()),('Paula', 'Johnson', 'paula@yahoo.com',
'123456', 'Massachusetts', 'sales', 0, now()),('Tom', 'Spears', 'tom@yahoo.com',
'123456', 'Massachusetts', 'sales', 0, now());


** SELECT **

# retrieve all columns and rows from table
```

```
SELECT * FROM users;

# retrieve only selected columns from table
SELECT first_name, last_name FROM users;

# return only unique values (no duplicates)
SELECT DISTINCT location FROM users;


** WHERE **

# filter rows by exact value match
SELECT * FROM users WHERE location='Massachusetts';

# filter rows with multiple conditions
SELECT * FROM users WHERE location='Massachusetts' AND dept='sales';

# filter rows where user is admin
SELECT * FROM users WHERE is_admin = 1;

# filter rows using comparison operator
SELECT * FROM users WHERE is_admin > 0;


** UPDATE **

# update existing row data based on condition
UPDATE users SET email = 'freddy@gmail.com' WHERE id = 2;


** DELETE **

# delete row permanently based on condition
DELETE FROM users WHERE id = 6;


** ALTER TABLE **

# add new column to existing table structure
ALTER TABLE users ADD age VARCHAR(3);

# modify existing column data type or definition
ALTER TABLE users MODIFY COLUMN age INT(3);


** ORDERING **

# sort results alphabetically ascending
SELECT * FROM users ORDER BY last_name ASC;

# sort results in descending order
SELECT * FROM users ORDER BY last_name DESC;


** STRING FUNCTIONS **

# combine columns into one readable output
SELECT CONCAT(first_name, ' ', last_name) AS 'Name', dept FROM users;

# convert text to upper/lower case
SELECT UCASE(first_name), LCASE(last_name) FROM users;


** RANGE **
```

```
# filter rows within numeric range
SELECT * FROM users WHERE age BETWEEN 20 AND 25;


** LIKE **

# match values starting with letter
SELECT * FROM users WHERE dept LIKE 'd%';

# match values starting with specific word
SELECT * FROM users WHERE dept LIKE 'dev%';

# match values ending with letter
SELECT * FROM users WHERE dept LIKE '%t';

# match values containing character
SELECT * FROM users WHERE dept LIKE '%e%';

# exclude matching pattern
SELECT * FROM users WHERE dept NOT LIKE 'd%';


** IN **

# filter rows matching any value from list
SELECT * FROM users WHERE dept IN ('design', 'sales');



** INDEX **

# create index to speed up search queries
CREATE INDEX LIndex ON users(location);

# remove index from table
DROP INDEX LIndex ON users;


** RELATIONS **

# create related table with foreign key reference
CREATE TABLE posts(
id INT AUTO_INCREMENT,
user_id INT,
title VARCHAR(100),
body TEXT,
publish_date DATETIME DEFAULT CURRENT_TIMESTAMP,
PRIMARY KEY(id),
FOREIGN KEY (user_id) REFERENCES users(id)
);

# insert record into related posts table
INSERT INTO posts(user_id, title, body) VALUES (1, 'Post One', 'This is post
one');

# create table with multiple foreign key relationships
CREATE TABLE comments(
id INT AUTO_INCREMENT,
post_id INT,
user_id INT,
body TEXT,
publish_date DATETIME DEFAULT CURRENT_TIMESTAMP,
PRIMARY KEY(id),
```

```
FOREIGN KEY(user_id) references users(id),
FOREIGN KEY(post_id) references posts(id)
);


** JOINS **

# combine user and post data using matching ids
SELECT users.first_name, users.last_name, posts.title, posts.publish_date
FROM users
INNER JOIN posts ON users.id = posts.user_id
ORDER BY posts.title;

# include all comments even if no matching post exists
SELECT comments.body, posts.title
FROM comments
LEFT JOIN posts ON posts.id = comments.post_id
ORDER BY posts.title;

# join multiple tables to combine related data
SELECT comments.body, posts.title, users.first_name, users.last_name
FROM comments
INNER JOIN posts on posts.id = comments.post_id
INNER JOIN users on users.id = comments.user_id
ORDER BY posts.title;


** AGGREGATE **

# count total number of rows
SELECT COUNT(id) FROM users;

# get maximum value from column
SELECT MAX(age) FROM users;

# get minimum value from column
SELECT MIN(age) FROM users;

# calculate sum of values
SELECT SUM(age) FROM users;


** GROUP BY **

# group rows and count occurrences per group
SELECT age, COUNT(age) FROM users GROUP BY age;

# group filtered rows only
SELECT age, COUNT(age) FROM users WHERE age > 20 GROUP BY age;

# group and filter aggregated results
SELECT age, COUNT(age) FROM users GROUP BY age HAVING count(age) >=2;


** BROWSING **

# show full create statement of table including schema and indexes
SHOW CREATE TABLE table;

# show all current running queries and connections
SHOW PROCESSLIST;

# kill a running query or connection by process id
KILL process_number;
```

```
** SELECT ADVANCED **

# select from multiple tables (cartesian product)
SELECT * FROM table1, table2;

# select specific fields from multiple tables
SELECT field1, field2 FROM table1, table2;

# select with condition
SELECT ... FROM ... WHERE condition;

# group results by field
SELECT ... FROM ... WHERE condition GROUP BY field;

# group and filter grouped results
SELECT ... FROM ... WHERE condition GROUP BY field HAVING condition2;

# sort results by multiple fields
SELECT ... FROM ... WHERE condition ORDER BY field1, field2;

# sort descending
SELECT ... FROM ... WHERE condition ORDER BY field1, field2 DESC;

# limit number of returned rows
SELECT ... FROM ... WHERE condition LIMIT 10;

# distinct multiple fields
SELECT DISTINCT field1, field2 FROM ...;


** JOINS ADVANCED **

# join two tables with condition
SELECT ... FROM t1 JOIN t2 ON t1.id1 = t2.id2 WHERE condition;

# left join keeps all rows from first table
SELECT ... FROM t1 LEFT JOIN t2 ON t1.id1 = t2.id2 WHERE condition;

# nested joins for multiple tables
SELECT ... FROM t1 JOIN (t2 JOIN t3 ON ...) ON ...;


** CONDITIONS **

# equals comparison
field1 = value1

# not equal comparison
field1 <> value1

# pattern matching with wildcards
field1 LIKE 'value _ %'

# check for null values
field1 IS NULL

# check for not null values
field1 IS NOT NULL

# match any value in list
field1 IS IN (value1, value2)
```

```
# exclude values from list
field1 IS NOT IN (value1, value2)

# logical AND condition
condition1 AND condition2

# logical OR condition
condition1 OR condition2


** DATABASE ADVANCED **

# create database with specific charset
CREATE DATABASE DatabaseName CHARACTER SET utf8;

# alter database charset
ALTER DATABASE DatabaseName CHARACTER SET utf8;


** BACKUP ADVANCED **

# backup database with custom user
mysqldump -u Username -p dbNameYouWant > databasename_backup.sql

# restore database from backup file
mysql -u Username -p dbNameYouWant < databasename_backup.sql;


** REPAIR **

# check and repair all databases after crash
mysqlcheck --all-databases;

# fast check only changed tables
mysqlcheck --all-databases --fast;


** INSERT ADVANCED **

# insert specific fields into table
INSERT INTO table1 (field1, field2) VALUES (value1, value2);


** DELETE ADVANCED **

# delete all rows from table
DELETE FROM table1;

# truncate table (fast delete all rows)
TRUNCATE table1;

# delete with condition
DELETE FROM table1 WHERE condition;

# delete using join condition
DELETE FROM table1, table2 WHERE table1.id1 = table2.id2 AND condition;


** UPDATE ADVANCED **

# update multiple tables with join
UPDATE table1, table2 SET field1=new_value1, field2=new_value2 WHERE table1.id1
= table2.id2 AND condition;
```

```
** CREATE TABLE ADVANCED **

# create simple table
CREATE TABLE table (field1 type1, field2 type2);

# create table with index
CREATE TABLE table (field1 type1, field2 type2, INDEX (field));

# create table with primary key
CREATE TABLE table (field1 type1, field2 type2, PRIMARY KEY (field1));

# create table with composite primary key
CREATE TABLE table (field1 type1, field2 type2, PRIMARY KEY (field1,field2));

# create table with foreign key constraint
CREATE TABLE table1 (fk_field1 type1, field2 type2, FOREIGN KEY (fk_field1)
REFERENCES table2 (t2_fieldA));

# create table if not exists
CREATE TABLE table IF NOT EXISTS;

# create temporary table (session only)
CREATE TEMPORARY TABLE table;


** DROP TABLE ADVANCED **

# drop table only if exists
DROP TABLE IF EXISTS table;

# drop multiple tables
DROP TABLE table1, table2;


** ALTER TABLE ADVANCED **

# change column type
ALTER TABLE table MODIFY field1 type1;

# rename column
ALTER TABLE table CHANGE old_name_field1 new_name_field1 type1;

# set default value
ALTER TABLE table ALTER field1 SET DEFAULT ...;

# remove default value
ALTER TABLE table ALTER field1 DROP DEFAULT;

# add column at first position
ALTER TABLE table ADD new_name_field1 type1 FIRST;

# add column after another column
ALTER TABLE table ADD new_name_field1 type1 AFTER another_field;

# remove column
ALTER TABLE table DROP field1;

# add index to table
ALTER TABLE table ADD INDEX (field);


** FIELD ORDER **
```

```
# move column to first position
ALTER TABLE table MODIFY field1 type1 FIRST;

# move column after another
ALTER TABLE table MODIFY field1 type1 AFTER another_field;


** KEYS **

# define composite primary key
CREATE TABLE table (..., PRIMARY KEY (field1, field2));

# define composite foreign key
CREATE TABLE table (..., FOREIGN KEY (field1, field2) REFERENCES table2
(t2_field1, t2_field2));


** USERS ADVANCED **

# create user without password
CREATE USER 'user'@'localhost';

# grant specific permissions only
GRANT SELECT, INSERT, DELETE ON base.* TO 'user'@'localhost' IDENTIFIED BY
'password';

# revoke privileges from user
REVOKE ALL PRIVILEGES ON base.* FROM 'user'@'host';

# change password for current user
SET PASSWORD = PASSWORD('new_pass');

# change password for specific user
SET PASSWORD FOR 'user'@'host' = PASSWORD('new_pass');


** DATA TYPES **

# integer types for storing whole numbers
TINYINT / SMALLINT / MEDIUMINT / INT / BIGINT

# floating point numbers
FLOAT / DOUBLE

# date and time types
DATE / DATETIME / TIMESTAMP / TIME / YEAR

# string types
VARCHAR / TEXT / BLOB

# enum for predefined values
ENUM ('value1', 'value2')


** RESET ROOT ALT **

# stop mysql service (older systems)
 /etc/init.d/mysql stop

# start mysql without permission checks
mysqld_safe --skip-grant-tables

# login without password
mysql
```

```
# update root password
UPDATE mysql.user SET password=PASSWORD('new_pass') WHERE user='root';

# start mysql normally
/etc/init.d/mysql start


** DATA TYPES **

# tiny integer for very small whole numbers (-128 to 127) used for
flags/booleans
TINYINT

# small integer for small range numbers (~ -65k to 65k)
SMALLINT

# medium integer for larger range numbers (~ ±16 million)
MEDIUMINT

# standard integer for most numeric values (~ ±2 billion)
INT

# large integer for very big numbers (~ ±9 quintillion)
BIGINT

# floating point number with decimal precision (less accurate, faster)
FLOAT

# double precision floating point (more accurate than float)
DOUBLE

# fixed date format (YYYY-MM-DD) used for storing calendar dates
DATE

# date and time format (YYYY-MM-DD HH:MM:SS)
DATETIME

# timestamp stored as unix time (auto-updates, used for logs)
TIMESTAMP

# time only format (HH:MM:SS)
TIME

# year value (YYYY)
YEAR

# variable-length string (single line text with max length)
VARCHAR(255)

# text field for large multi-line strings
TEXT

# binary large object for storing files like images
BLOB

# very small text (max 255 chars)
TINYTEXT

# medium size text (~16MB)
MEDIUMTEXT

# very large text (~4GB)
LONGTEXT
```

```
# enum restricts column to predefined values
ENUM('value1','value2')

# boolean-like tinyint (0=false, 1=true)
TINYINT(1)

# decimal exact numeric type (precise financial calculations)
DECIMAL(10,2)
```