MariaDB
Database Administration

MySQL and MariaDB history

- MySQL founded 1995
- acquired by Sun Microsystems 2008
- now owned by Oracle Corporation since 2009
- soon after MariaDB started by original authors of MySQL

Release Background

- 1994 - MySQL by Michael (Monty) Widenius and David Axmark
- First internal release on 23 May 1995
- Windows version was released on 8 January 1998 for Windows 95 and NT
- Version 3.23 January 2001
- Version 4.0 March 2003 (unions)
- Version 4.1 October 2004 (R-trees and B-trees, subqueries, prepared statements)
- Version 5.0 October 2005 (cursors, stored procedures, triggers, views)
- Sun Microsystems acquired MySQL AB on 26 February 2008.

Release Background (contd.)

- Version 5.1 November 2008 (event scheduler, partitioning, plugin API, row-based replication, server log tables)
- MySQL 5.1 and 6.0 showed poor performance when used for data warehousing — partly due to its inability to utilize multiple CPU cores for processing a single query.[26]
- MariaDB initial release - 22 January 2009
- Oracle acquired Sun Microsystems on 27 January 2010. Oracle and Sun
- December 2012 - MariaDB foundation has been established
- On October 1, 2014, SkySQL Corporation Ab changed its name to MariaDB Corporation Ab

MariaDB Server Installation

- Install the server discussing different options
- GUI Tools Instalation
- Quick Overview on GUI Tools

Installing in Ubuntu

- https://downloads.mariadb.org/mariadb/repositories/#mirror=coreix

Starting and stopping server

- Linux
  - `sudo service mysql start`
  - `sudo service mysql stop`

- Windows (as a serverice)
  - `net start mysql`

- MariaDB 10.0 starting from command line
  - `mysqld --console`

Installing on Windows

- https://downloads.mariadb.org/
- Follow installer steps

Hardware and MariaDB

- A lot of MariaDB features has been created when memory and CPU power was expensive and not flexible
- Avoid things specific to MariaDB
- For example, the physical hard drive failure is not an issue with cloud computing
- As for partitioning, it can be handle by the storage itself (S3, RAID, etc...) Usually you can buy it very cheaply
- Optimization vs hardware, it is usually cheaper to buy extra resources, more complicated optimization is the last resort

MySQL and Platform

- Using Debian, Ubuntu or other Linux distribution, you don't have to do upgrade and huge chunk of maintenance work yourself
- Windows 32bit memory limitation is 3.2 GB, around 60% of optimization problems was related to this limitation
- Changing default behaviour means that the changes must be documented or automated during the application installation process. This can depend on the platform as well.
- The most suitable platforms are Linux and Windows. Other systems (despite vendors promises) do not have so big community support
- Choosing platform is important and does effect the effort needed to maintain the system

MariaDB and OS

- Some configuration options of the server depend of the platform.
- Using frameworks and database separation libraries (persistence), like Spring, Drupal, etc... can avoid common mistakes
- If your database is too big and you "think" you need new features, look at the data from "business" perspective.
  - Is it redundant?
  - Does it have to be stored in the main database?
  - Can we delete or move old data?
- Usually good data design and cleansing process solves 80% of cases I encountered. Using partitioning, replication or cluster is much more expensive compare to business like solution.
- Smart SOA allows you to avoid big monolithic applications with huge databases

MariaDB Storage Engines

- MariaDB supports several storage engines that act as handlers for different table types.
- Some table are transaction-safe and nontransaction-safe

- `SHOW ENGINES;`
  - Shows supported storage engine

MySQL Storage Engines

InnoDB/XtraDB

- A transaction-safe (ACID compliant)
- Has commit, rollback, and crash-recovery capabilities
- InnoDB row-level locking and Oracle-style consistent non-locking reads
- Good for increase multi-user concurrency and performance
- InnoDB stores user data in clustered indexes to reduce I/O for common queries based on primary keys
- Supports FOREIGN KEY referential-integrity constraints

MySQL Storage Engines

MyISAM

- Mainly for web and data warehouses
- Doesn’t support transactions and foreign keys

Memory (HEAP)

- Stores all data in RAM for extremely fast access in environments that require quick lookups of reference and other like data

Merge

- Logically groups a series of identical MyISAM tables and reference them as one object
- Good for VLDB environments such as data warehousing

MySQL Storage Engines

Archive

- Provides the perfect solution for storing and retrieving large amounts of seldom-referenced historical, archived, or security audit information.

CSV

- Stores data in text files using comma-separated values format.
- Easily exchange data between other software and applications that can import and export in CSV format
- Use to store log files

Blackhole

- accepts but does not store data and retrievals always return an empty set
- Can be used in distributed database design where data is automatically replicated, but not stored locally.

MyISAM vs. InnoDB

MyISAM	InnoDB
Few DML operations	OLTP with a lot of DML
Very fast for selects	Slower for selects
Some data can get corrupted	Good crash-recovery facility
Cannot be a MASTER server in replication, but can be a SLAVE	Perfect for MASTER server in replication
	Supports FOREIGN KEY
	Binary logs
	Point-in-time recovery

MyISAM vs. InnoDB

Feature	MyISAM	innoDB
Storage limits	256TB	64TB
Transactions	No	Yes
Locking granularity	Table	Row
MVCC	No	Yes
Geospatial indexing	Yes	No
Full-text search indexes	Yes	No
Clustered indexes	No	Yes
Data caches	No	Yes
Cluster database support	No	No
Foreign key support	No	Yes
Point-in-time recovery	No	Yes

MariaDB Server Files and Scripts

- MariaDB Programs
- Server Programms
- Client Programms
- GUI and other tools

MariaDB Programs

- SERVER
  - mysqld - the MySQL server
  - mysqld_safe - a script which starts mysqld, recommended of starting the server
    - restarts the server when an error occur
    - logs run-time and failure information
  - mysqld_multi - script that can start or stop multiple servers installed on the
- system (different ports or sockets)

MariaDB Programs

- CLIENT
  - mysql - a command line client
  - mysqladmin - administrative operations
    - reloading the grant tables
    - creating/dropping database
    - flushing tables to disk
    - managing log files
    - shutdown the database

MariaDB Programs

- MariaDB Database
  - Visual database design application
- MariaDB Galera Cluster
  - Database Cluster
- Client Libraries (Java/C, etc..) and ODBC Drivers
  - Connectivity from applications
- Graphical Clients
  - https://mariadb.com/kb/en/mariadb/documentation/clients-and-utilities/graphical-clients/

Ways of starting a server

- Windows or Unix service
  - `sudo /etc/init.d/mysql start`
- Mysqld_safe
  - `sudo mysqld_safe`
- Command line mysqld (Unix)
  - `sudo mysqld`
- Command line Windows (MariaDB)
  - `mysqld --console`

Stopping MariaDB Server

- Windows/Unix Service
  - sudo /etc/init.d/mysql stop
  - sudo service mysql stop
  - net stop mysql (Windows)
- mysqladmin
  - mysqladmin -u root -p shutdown

Changing Forgotten Password

- Run
  - `mysqld --skip-grant-tables  --skip-networking`
  - This option causes the server to start without using the privilege system at all, which gives anyone with access to the server unrestricted access to all databases
- Executing
  - `mysqladmin flush-privileges` or
  - `mysqladmin reload` or
  - `mysqladmin FLUSH PRIVILEGES`
- Change your password
  - `set password for root@localhost = password('asdfasdf')`

Running Scripts

- From MySQL prompt
  - `use test`
  - `source <path-to-file>`
- From Command Line (Unix-like os)
  - `cat filename | mysql -uroot -pWelcome#123`
- From Command Line (Windows)
  - `type filename | mysql -uroot -pWelcome#123`

- Also:
  - `echo ‘select * from EMP’ | mysql -uroot -p mydb1`
  - `mysql -uroot -pWelcome#123 mydb1 < script1.sql`

MariaDB Server Configuration

- mysqld Options
- Server System Variables
- Dynamic System Variables
- Server Status Variables
- The Server SQL Mode
- Shutdown Process

mysqld server options

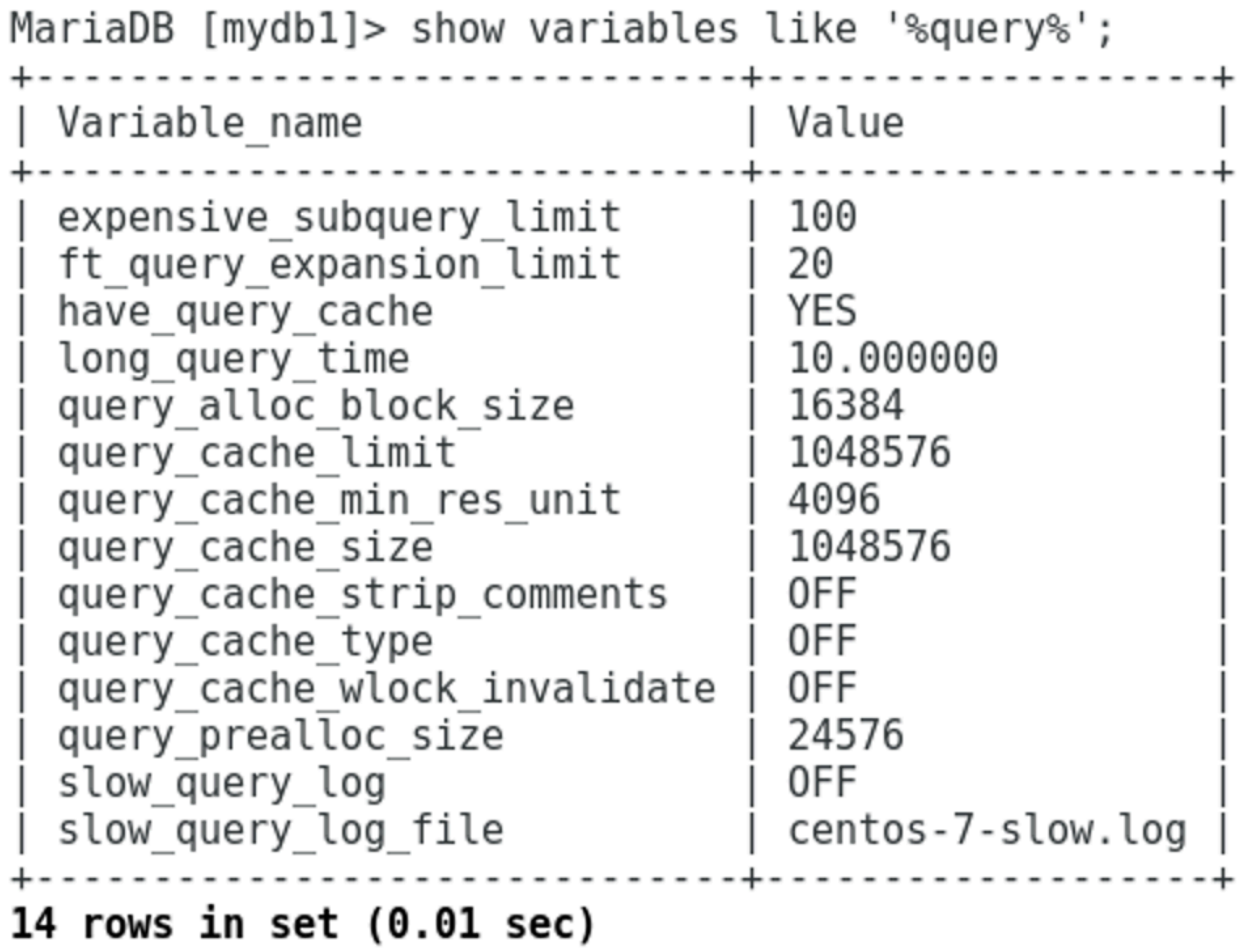
- MariaDB Server manages access to the MySQL data directory that contains databases and tables.
- For a complete list of options, run this command:
  - `mysqld --verbose --help`

Passing options to MariaDB server

- Cmd-Line
- Options file
- System Var
- Status Var [session or global or both]

Checking variable’s value

- `show variables like ‘%query%'`



```
MariaDB [mydb1]> show variables like '%query%';
+------------------------------+-------------------+
| Variable_name                | Value             |
+------------------------------+-------------------+
| expensive_subquery_limit     | 100               |
| ft_query_expansion_limit     | 20                |
| have_query_cache             | YES               |
| long_query_time              | 10.000000         |
| query_alloc_block_size       | 16384             |
| query_cache_limit            | 1048576           |
| query_cache_min_res_unit     | 4096              |
| query_cache_size             | 1048576           |
| query_cache_strip_comments   | OFF               |
| query_cache_type             | OFF               |
| query_cache_wlock_invalidate | OFF               |
| query_prealloc_size          | 24576             |
| slow_query_log               | OFF               |
| slow_query_log_file          | centos-7-slow.log |
+------------------------------+-------------------+
14 rows in set (0.01 sec)
```

Parameters Formats

- Command Line Format
  - `mysqld --sync-binlog=0`
- Config File Format
  - `sync-binlog`
- Variable Name
  - `sync_binlog`

Restricting maximum value of a variable ⌘

- `mysqld --maximum-`***`var_name`***`=value`

- For example, to prevent the value of query_cache_size from being increased to more than 32MB at runtime, use the option —maximum-query_cache_size=32M

Server Options for Loading Plugins

- Enable the plugin
- `--plugin_name[=ON]`
- `--plugin_name=1`
- `--enable-plugin_name`

Server Options for Loading Plugins

- If plugin initialization fails, start the server anyway, but with the plugin disabled
  - `--plugin_name=FORCE`
  - If plugin initialization fails, do not start the server

- Disable plugin
  - `--plugin_name=OFF`
  - `--skip-plugin_name`
  - `--pluin_name=0`
  - `--disable-plugin_name`

System variables

- indicate how the server is configured
- accessible via "`Select @@variablename`"

Session Variables

- Cannot be set at server startup

Status Variable

- `show status;`

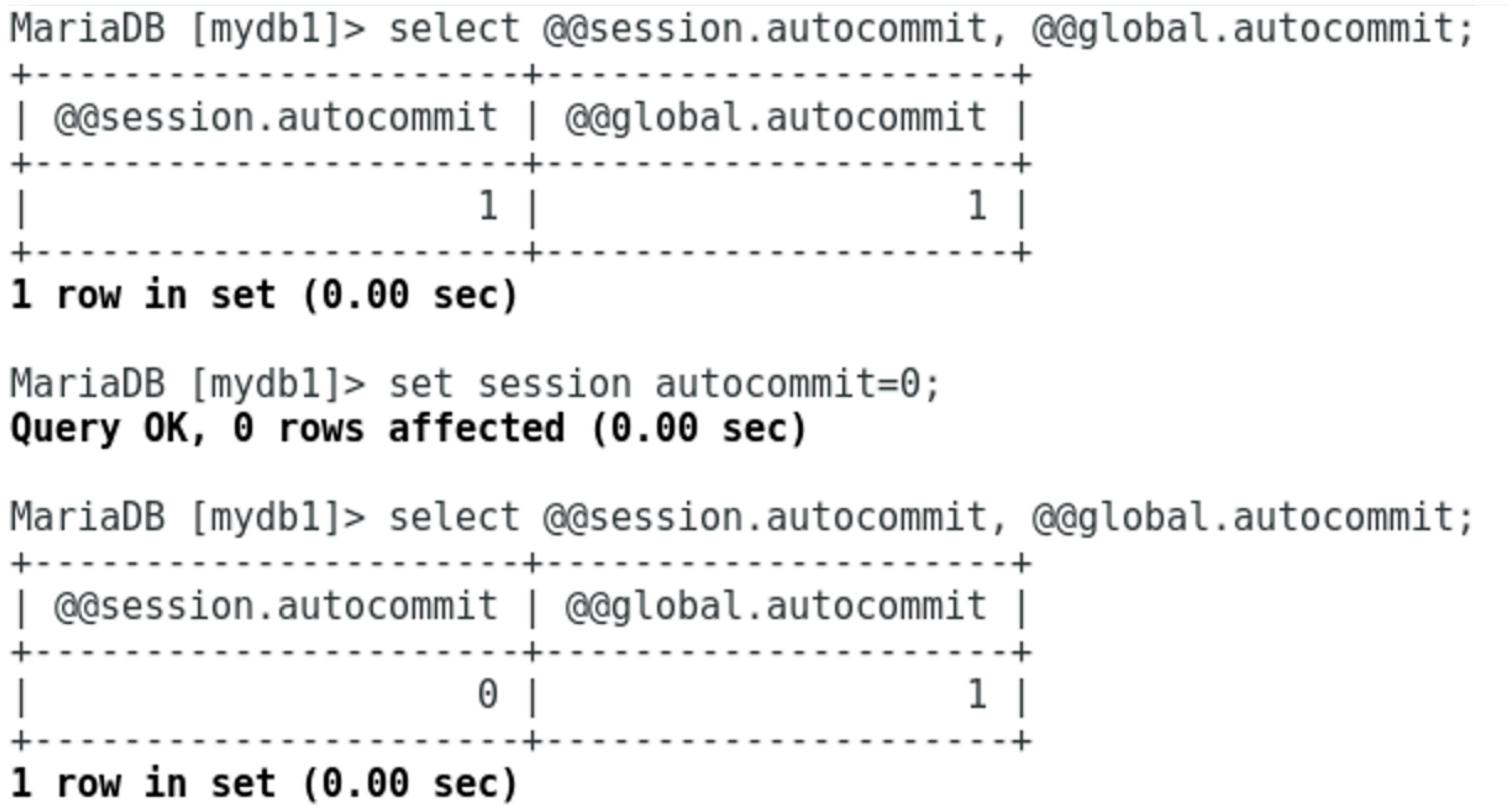
Scope of the variable

- GLOBAL
- SESSION

Session System Variables

- exist only as session variables
- these cannot be set at server startup
- can be assigned values at runtime using the SET statement (except for those that are read only)
- Most of them are not displayed by SHOW VARIABLES, but you can obtain their values using SELECT.

Session System Variables



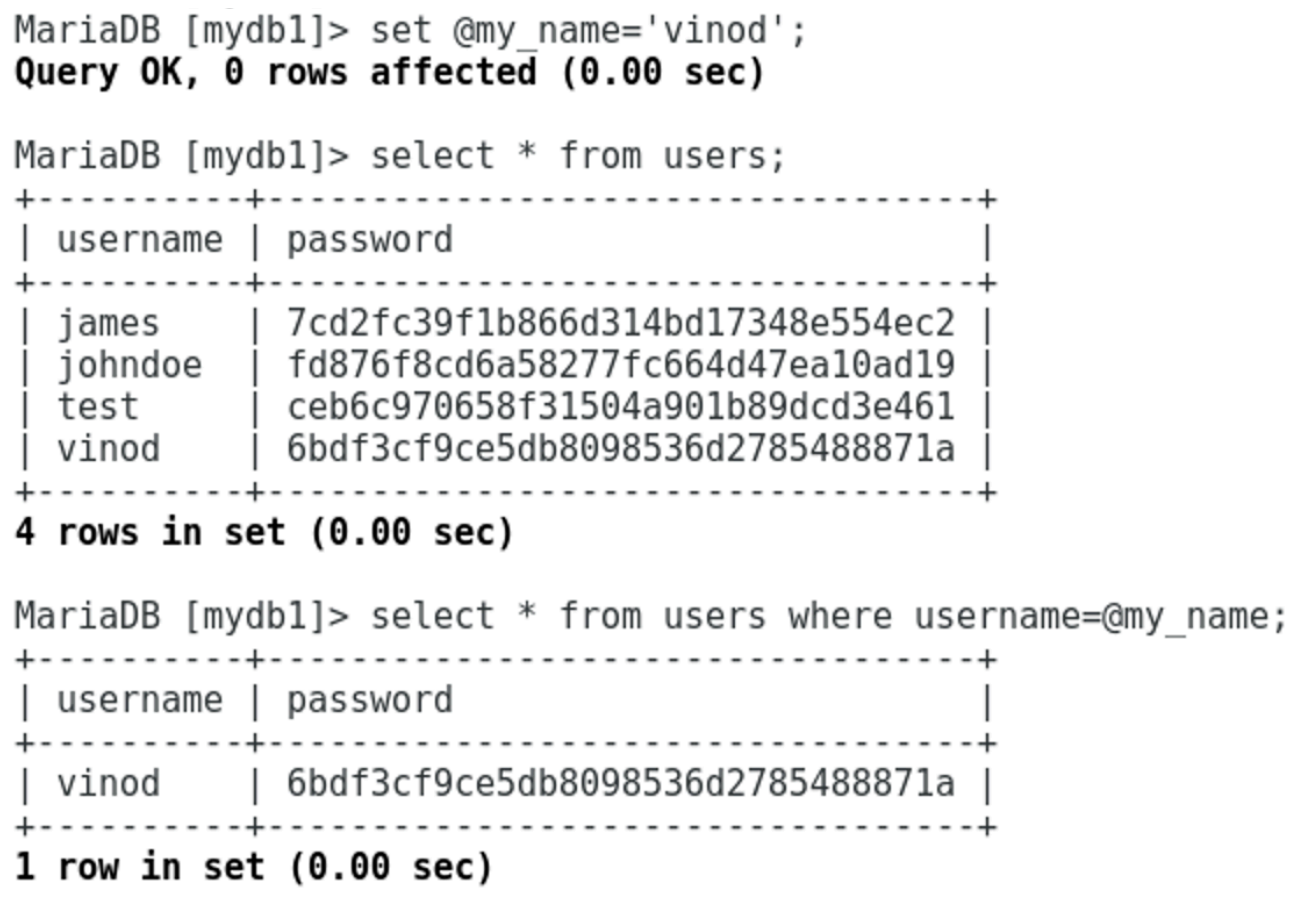
```
MariaDB [mydb1]> select @@session.autocommit, @@global.autocommit;
+----------------------+---------------------+
| @@session.autocommit | @@global.autocommit |
+----------------------+---------------------+
|                    1 |                   1 |
+----------------------+---------------------+
1 row in set (0.00 sec)

MariaDB [mydb1]> set session autocommit=0;
Query OK, 0 rows affected (0.00 sec)

MariaDB [mydb1]> select @@session.autocommit, @@global.autocommit;
+----------------------+---------------------+
| @@session.autocommit | @@global.autocommit |
+----------------------+---------------------+
|                    0 |                   1 |
+----------------------+---------------------+
1 row in set (0.00 sec)
```

Server Status Variables

- Status variables provide information about its operation
  - `SHOW [GLOBAL | SESSION] STATUS`
- GLOBAL keyword aggregates the values over all connections
- SESSION shows the values for the current connection.
- FLUSH STATUS statement restarts the status variables
- Cannot be access like system variables via select statement "select @@tmpdir"
  - `SHOW GLOBAL STATUS;`
  - `SHOW SESSION STATUS LIKE ‘wsrep%’;`

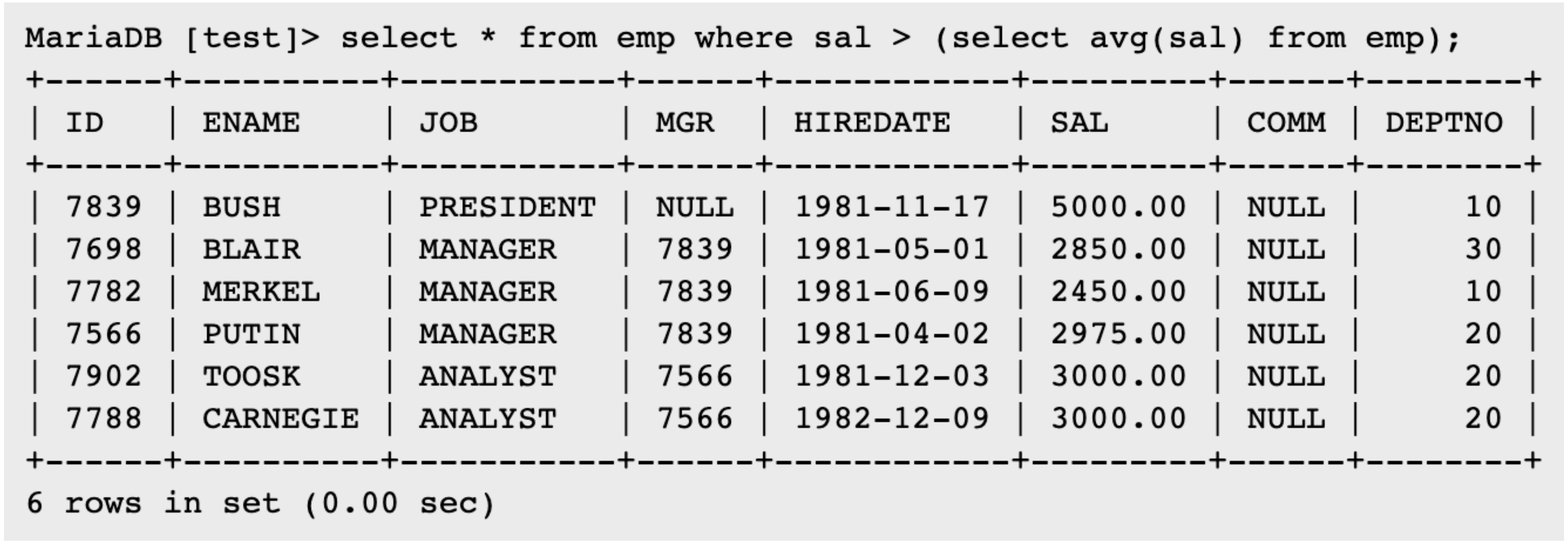
Local (non system) Variables

- Single "at charater"
- Visible only in the session
- Useful with SQL scripts



```
MariaDB [mydb1]> set @my_name='vinod';
Query OK, 0 rows affected (0.00 sec)

MariaDB [mydb1]> select * from users;
+----------+----------------------------------+
| username | password                         |
+----------+----------------------------------+
| james    | 7cd2fc39f1b866d314bd17348e554ec2 |
| johndoe  | fd876f8cd6a58277fc664d47ea10ad19 |
| test     | ceb6c970658f31504a901b89dcd3e461 |
| vinod    | 6bdf3cf9ce5db8098536d2785488871a |
+----------+----------------------------------+
4 rows in set (0.00 sec)

MariaDB [mydb1]> select * from users where username=@my_name;
+----------+----------------------------------+
| username | password                         |
+----------+----------------------------------+
| vinod    | 6bdf3cf9ce5db8098536d2785488871a |
+----------+----------------------------------+
1 row in set (0.00 sec)
```

Using Local Variables to replace subqueries



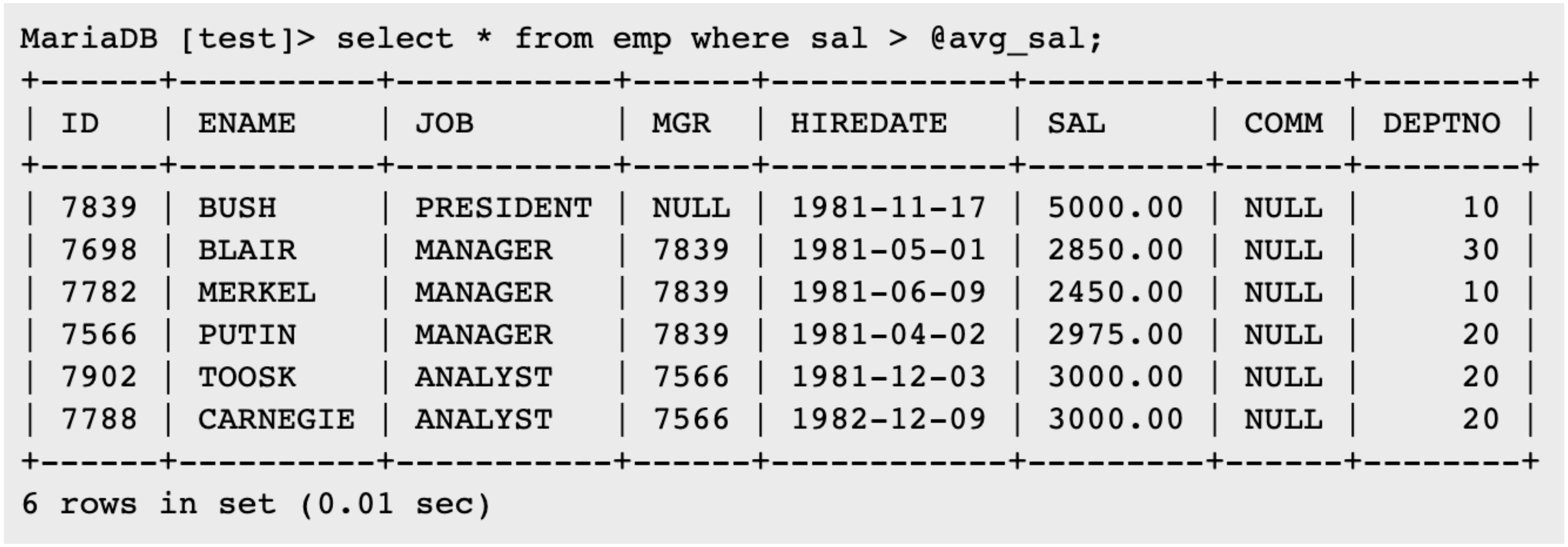
```
MariaDB [test]> select * from emp where sal > (select avg(sal) from emp);
+------+----------+-----------+------+------------+---------+------+--------+
| ID   | ENAME    | JOB       | MGR  | HIREDATE   | SAL     | COMM | DEPTNO |
+------+----------+-----------+------+------------+---------+------+--------+
| 7839 | BUSH     | PRESIDENT | NULL | 1981-11-17 | 5000.00 | NULL |     10 |
| 7698 | BLAIR    | MANAGER   | 7839 | 1981-05-01 | 2850.00 | NULL |     30 |
| 7782 | MERKEL   | MANAGER   | 7839 | 1981-06-09 | 2450.00 | NULL |     10 |
| 7566 | PUTIN    | MANAGER   | 7839 | 1981-04-02 | 2975.00 | NULL |     20 |
| 7902 | TOOSK    | ANALYST   | 7566 | 1981-12-03 | 3000.00 | NULL |     20 |
| 7788 | CARNEGIE | ANALYST   | 7566 | 1982-12-09 | 3000.00 | NULL |     20 |
+------+----------+-----------+------+------------+---------+------+--------+
6 rows in set (0.00 sec)
```

Using Local Variables to replace subqueries

```
MariaDB [test]> select avg(sal) from emp;
+-------------+
| avg(sal)    |
+-------------+
| 2073.214286 |
+-------------+
1 row in set (0.00 sec)
```

```
MariaDB [test]> select avg(sal) from emp into @avg_sal;
Query OK, 1 row affected (0.00 sec)
```

Using Local Variables to replace subqueries



```
MariaDB [test]> select * from emp where sal > @avg_sal;
+------+----------+-----------+------+------------+---------+------+--------+
| ID   | ENAME    | JOB       | MGR  | HIREDATE   | SAL     | COMM | DEPTNO |
+------+----------+-----------+------+------------+---------+------+--------+
| 7839 | BUSH     | PRESIDENT | NULL | 1981-11-17 | 5000.00 | NULL |     10 |
| 7698 | BLAIR    | MANAGER   | 7839 | 1981-05-01 | 2850.00 | NULL |     30 |
| 7782 | MERKEL   | MANAGER   | 7839 | 1981-06-09 | 2450.00 | NULL |     10 |
| 7566 | PUTIN    | MANAGER   | 7839 | 1981-04-02 | 2975.00 | NULL |     20 |
| 7902 | TOOSK    | ANALYST   | 7566 | 1981-12-03 | 3000.00 | NULL |     20 |
| 7788 | CARNEGIE | ANALYST   | 7566 | 1982-12-09 | 3000.00 | NULL |     20 |
+------+----------+-----------+------+------------+---------+------+--------+
6 rows in set (0.01 sec)
```

Server SQL Modes

- An SQL mode changes the syntax of the SQL and some client behaviours
- Modes can apply differently for different clients
- Application written to a different SQL dialects can be used or migrated with less effort
- The default value is empty (no modes set)
- SET [GLOBAL|SESSION] sql_mode='modes'
- SELECT @@GLOBAL.sql_mode;
- SELECT @@SESSION.sql_mode;
- ANSI, DB2, MSSQL, ORACLE, POSTGRESQL, TRADITIONAL
- More info:
  - http://dev.mysql.com/doc/refman/5.7/en/sql-mode.html

The Shutdown Process

- The shutdown process can be initiated several ways.
  - execute a `mysqladmin shutdown` command.
  - system-specific methods: Unix - SIGTERM signal, Windows - services manager
- The server might create a thread to handle the shutdown process if necessary.
- The server stops accepting new connections to prevent new activity from being initiated during shutdown
- The server terminates current activity
  - open transactions are rolled back
  - non-transactional table, multiple-row UPDATE or INSERT may leave the table partially updated
- Storage engines are shut down or closed.
  - table cache is flushed
  - all open tables are closed.
- The server exits.

MariaDB
Security and Privilege System

Privilege System Overview

- MariaDB privilege system:
  - Authenticates a user who connects from a given host
  - Associates that user with privileges on a database such as SELECT, INSERT, UPDATE, and DELETE.
- Anonymous user handling is deprecated and will be skipped in this presentation.

CHECKING USER PRIVILEGES

- show grants for a specific account
  - `SHOW GRANTS FOR vinod@localhost;`
- Show grants for a current account
  - `SHOW GRANTS;`

What you cannot do with the privilege system

- You cannot explicitly specify that a given user should be denied access.
- You cannot specify that a user has privileges to create or drop tables in a database but not to create or drop the database itself.
- A password applies globally to an account. You cannot associate a password with a specific object such as a database, table, or routine.

MariaDB Permission System

- There are only three statements CREATE USER, GRANT, and REVOKE
- privilege information are stored in the grant tables of the mysql database
- server reads the contents of these tables into memory when it starts and bases access-control decisions on the in-memory copies of the grant tables (you can synchronize them with flush statement)
- The MySQL privilege system ensures that all users may perform only the operations allowed to them
- During connection your identity is determined by:
  - the host from which you connect
  - the user name you specify

Users and hosts

- The same username can have totally different permission depend on the host they connect from:
  - `SHOW GRANTS FOR ’vinod’@’office.example.com’;`
  - `SHOW GRANTS FOR ’vinod’@’home.example.com’;`

Permission Control Stages

- Stage 1 (Authentication, Connection Verification)
  - The server accepts or rejects the connection based on your identity and whether you can verify your identity by supplying the correct password.
- Stage 2 (Authorization, Request Verification)
  - Assuming that you can connect, the server checks each statement you issue to determine whether you have sufficient privileges to perform it.

MySQL Privileges

- Administrative
  - enable users to manage operation of the MySQL server.
  - are global, not specific to a particular database.
- Database
  - apply to a database and to all objects within it
  - can be granted for specific databases, or globally so that they apply to all databases.
- Privileges for database objects such as tables, indexes, etc.. can be granted:
  - for specific objects within a database,
  - for all objects of a given type within a database (for example, all tables in a database)
  - globally for all objects of a given type in all databases).

Detailed Privileged

- ALL or ALL PRIVILEGES specifier is shorthand for “all privileges available at a given privilege level” (except GRANT OPTION)
  - For example, granting ALL at the global or table level grants all global privileges or all table-level privileges.
  - http://dev.mysql.com/doc/refman/5.7/en/privileges-provided.html

Showing All Users

- `select distinct concat(user,’@’,host) from mysql.user;`

Specifying Account Names

- appears in CREATE USER, GRANT, SET PASSWORD
  - `'user_name'@'host_name'`
- ’me’ is equivalent to ’me’@’%’
- Quotes are necessary to specify:
  - a user_name string containing special characters (such as “-”),
  - a host_name string containing special characters or wildcard characters (such as “%”); for example, ’test-user’@’%.com’
- backticks (“`”), single quotes (“’”), or double quotes (“"”) can be used
- Write ’me’@’localhost’, not ’me@localhost’; the latter is interpreted as ’me@localhost’@’%’
- The anonymous account is @’localhost’.

Specifying Host

- Host name or an IP number ’localhost’,’127.0.0.1’
- Wildcard characters
  - `“%” and “_”, ’%.mysql.com’ ,’192.168.1.%’`
- Someone could try to exploit this capability by naming a host 192.168.1.somewhere.com
- To foil such attempts, host names that start with digits and a dot are not valid. Hosts like 1.2.example.com never matches the host part of account names. An IP wildcard value can match only IP numbers, not host names
- host_ip/netmask,’david’@’192.58.197.0/255.255.255.0’
- client_ip & netmask = host_ip
- Only A, B or C network classes are valid
- The following netmask (28 bits) will not work 192.168.0.1/255.255.255.240

Multiple Matches

- When multiple matches are possible, the server must determine which of them to use.
- It resolves this issue as follows:
  - Whenever the server reads the user table into memory, it sorts the rows.
  - When a client attempts to connect, the server looks through the rows in sorted order.
  - The server uses the first row that matches the client host name and user name.
  - It orders the rows with the most-specific Host values first. Literal host names and IP addresses are the most specific.
  - Anonymous user is the least specific user, therefore is the last match

When Privilege Changes Take Effect

- When MariaDB starts, it reads all grant table contents into memory.
- GRANT, REVOKE, or SET PASSWORD, the server loads the grant tables into memory again immediately.
- If you modify the grant tables directly using statements such as INSERT, UPDATE, or DELETE, your changes have no effect on privilege checking until you either restart the server or tell it to reload the tables.
- Flushing privileges
    - `FLUSH PRIVILEGES`
    - `mysqladmin flush-privileges`
    - `mysqladmin reload`

When Privilege Changes Take Effect

- When the server reloads the grant tables, privileges for each existing client connection are affected as follows:
- Table and column privilege changes take effect with the client’s next request
- Database privilege changes take effect the next time the client executes a USE db_name statement
- Global privileges and passwords are unaffected for a connected client. These changes take effect only for subsequent connections.
- If the server runs with --skip-grant-tables option, it does not read the grant tables or implement any access control. Anyone can *connect and do anything. Flush the privileges to enforce privileges

Making privileges simple

- If not sure use CURRENT_USER() to determine which row has been matched

ACCOUNT MANAGEMENT

MariaDB User Account Management

- Users and Passwords
- Creating New Users
- Deleting User Accounts
- Limiting User Resources
- Changing Passwords

User Names and Passwords ⌘

- Account is made of:
  - user name
  - client host or hosts
  - password
- Most MySQL clients by default try to log in using the current Unix user name as the MySQL user name
- MySQL user names can be up to 16 characters long
- MySQL encrypts passwords using its own algorithm - PASSWORD() SQL function
- Differs from the Unix password - see ENCRYPT()
  - `mysql --user=vinod --password=Welcome#123 vin_db`
  - `mysql -uvinod -pWelcome#123 vin_db`

Adding User Accounts

- You can create MySQL accounts in the following ways:
  - CREATE USER or GRANT. These statements cause the server to make appropriate modifications to the grant tables (preferred)
  - By manipulating the MySQL grant tables directly with statements such as INSERT, UPDATE, or DELETE
  - GUI or other tools (MySQL Admin, phpMyAdmin)

User Creation Examples

- GRANT ALL ON *.* TO ’vinod’@’localhost’ IDENTIFIED BY ’Welcome#123’;

Is equivalent to:

- CREATE USER vinod@localhost;
- GRANT ALL ON *.* TO vinod@localhost
- SET PASSWORD FOR localhost=PASSWORD(’Welcome#123’);

Manual User Creation

- INSERT INTO user VALUES('%', 'vinod',PASSWORD('some_pass'), 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', , , , , 0, 0, 0, 0);

OR

- INSERT INTO user SET Host='localhost', User='vinod', Reload_priv='Y', Process_priv=‘Y';

OR

- INSERT INTO user (Host,User,Password) VALUES('localhost', 'vinod',);

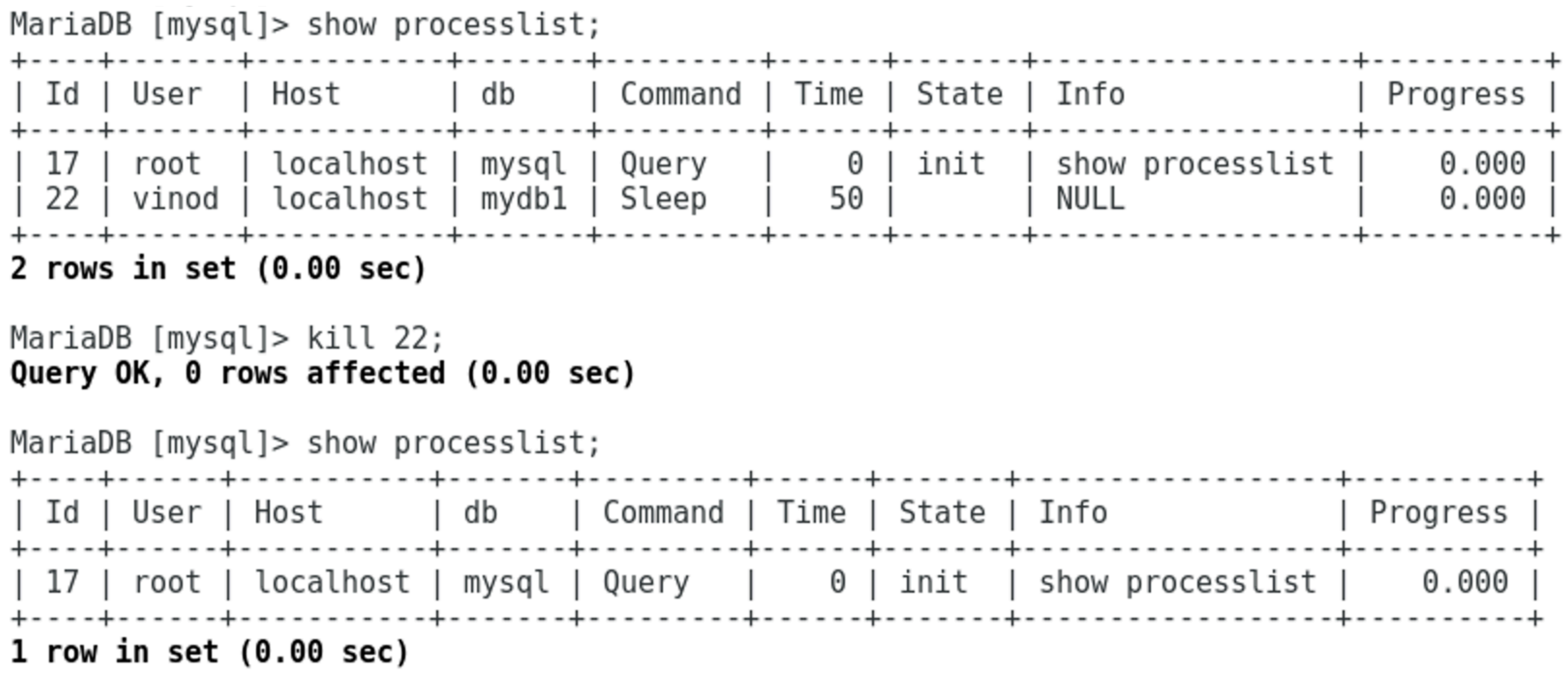
Also, execute the following:

- FLUSH PRIVILEGES;

Removing Users

- DROP USER user [, user] ..
- Each user@host pair should be removed
- DROP USER user is equivalent to DROP USER user@'%'
- DROP USER does not automatically close any open user sessions
- DROP USER does not automatically delete or invalidate any database objects that the user created.

Killing User Sessions



```
MariaDB [mysql]> show processlist;
+----+-------+-----------+-------+---------+------+-------+------------------+----------+
| Id | User  | Host      | db    | Command | Time | State | Info             | Progress |
+----+-------+-----------+-------+---------+------+-------+------------------+----------+
| 17 | root  | localhost | mysql | Query   |    0 | init  | show processlist |    0.000 |
| 22 | vinod | localhost | mydb1 | Sleep   |   50 |       | NULL             |    0.000 |
+----+-------+-----------+-------+---------+------+-------+------------------+----------+
2 rows in set (0.00 sec)

MariaDB [mysql]> kill 22;
Query OK, 0 rows affected (0.00 sec)

MariaDB [mysql]> show processlist;
+----+------+-----------+-------+---------+------+-------+------------------+----------+
| Id | User | Host      | db    | Command | Time | State | Info             | Progress |
+----+------+-----------+-------+---------+------+-------+------------------+----------+
| 17 | root | localhost | mysql | Query   |    0 | init  | show processlist |    0.000 |
+----+------+-----------+-------+---------+------+-------+------------------+----------+
1 row in set (0.00 sec)
```

Modes and User Creation

- Some of the statements will fail if the server's SQL mode has been set to enable certain restrictions.
  - strict mode (STRICT_TRANS_TABLES, STRICT_ALL_TABLES)
  - NO_AUTO_CREATE_USER
- will prevent the server from accepting some of the statements.

Changing Password ⌘

- During user creation
  - `CREATE USER 'vinod'@'localhost' IDENTIFIED BY 'secret';`
- Administrators for different user
  - `SET PASSWORD FOR 'vinod'@'localhost' = PASSWORD('secret');`
- Users themselves
  - `SET PASSWORD = PASSWORD('secret');`
  - `GRANT USAGE ON *.* TO 'vinod'@'localhost' IDENTIFIED BY 'secret';`
- From command line
  - `mysqladmin -uroot -pWelcome#123 password secret`

Common User Creation Scenarios

- GRANT ALL ON test.* TO user@localhost IDENTIFIED BY 'password' ;
  - Creates an user@localhost account if it doesn't exists
  - Grants ALL PRIVILEGES on all tables in the test database, even those create after the command was executed.

Limiting Account Resources

- max_user_connections system variable limits only the number of simultaneous connections made using a single account
- Individual accounts can have following limits:
  - The number of queries that an account can issue per hour
  - The number of updates that an account can issue per hour
  - The number of times an account can connect to the server per hour
  - The number of simultaneous connections to the server an account can have

Limiting Resources

```
CREATE USER 'vinod'@'localhost' IDENTIFIED BY ‘secret';

GRANT ALL ON customer.* TO 'vinod'@'localhost'
    WITH
    MAX_QUERIES_PER_HOUR  2
    MAX_UPDATES_PER_HOUR 10
    MAX_CONNECTIONS_PER_HOUR 5
    MAX_USER_CONNECTIONS 2;
```

Limiting Resources

- Modifying existing accounts:

```
GRANT USAGE ON *.* TO 'vinod'@'localhost'
   WITH MAX_QUERIES_PER_HOUR 100;
```

- To remove an existing limit, set its value to zero.

Limiting Resources

- To reset the current counts to zero for all accounts, issue a FLUSH USER_RESOURCES statement.
- The counts also can be reset by reloading the grant tables (for example, with a FLUSH PRIVILEGES statement or a mysqladmin reload command).
- Counter resets do not affect the MAX_USER_CONNECTIONS limit.
- All counts begin at zero when the server starts; counts are not carried over through a restart.

Common Pitfall

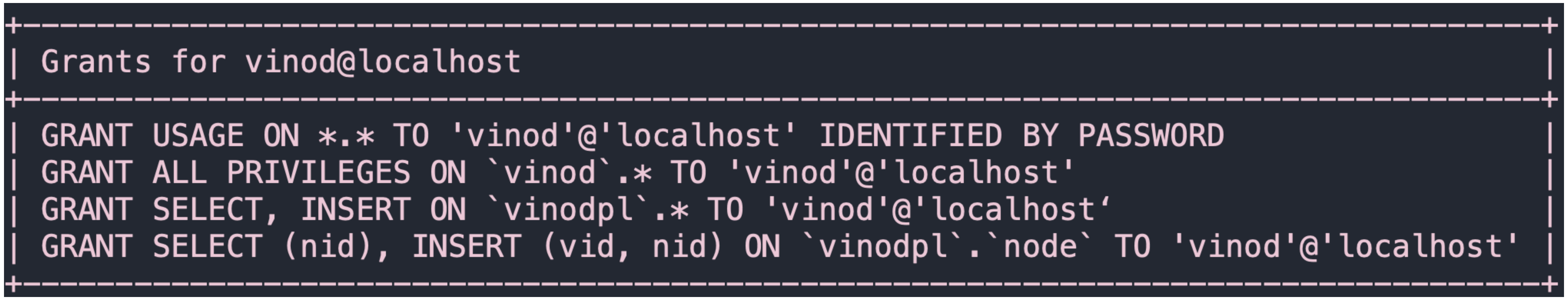
- Create two accounts in the following order: user@localhost and user@127.0.0.0/255.255.255.0
- Grant the users full permission on test table
- Login as the user from the local host (mysql –u user)
- Check which user you are logged in: select current_user();
- Drop both accounts and recreate them in the reverse order
- Follow the step 3 and 4, is the result the same?

Column Privileges

- `grant select (name,sal), insert (sal) on NP.emp to test@localhost;`

Global, Database, Column Privileges

- `SHOW GRANTS FOR vinod@localhost;`



```
+--------------------------------------------------------------------------------------+
| Grants for vinod@localhost                                                           |
+--------------------------------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'vinod'@'localhost' IDENTIFIED BY PASSWORD                     |
| GRANT ALL PRIVILEGES ON `vinod`.* TO 'vinod'@'localhost'                             |
| GRANT SELECT, INSERT ON `vinodpl`.* TO 'vinod'@'localhost‘                           |
| GRANT SELECT (nid), INSERT (vid, nid) ON `vinodpl`.`node` TO 'vinod'@'localhost' |
+--------------------------------------------------------------------------------------+
```

MariaDB
Database Maintenance

Backup and Recovery

- Logical versus physical backups
- Online versus offline backups
- Local versus remote backups
- Snapshot backups
- Full versus incremental backups
- Point-in-time recovery
- Backup scheduling, compression, and encryption
- Table maintenance

Logical versus physical backups

- Logical backups
  - CREATE DATABASE, CREATE TABLE statements and content (INSERT statements, text files or XML).
- Physical backups
  - raw copies of the directories and files

Logical vs. Physical

- Logical
  - data pass MySQL server
  - is much slower than physical
  - output is usually larger
  - backup and restore granularity (all databases, database, specific tables)
  - does not include log or configuration files
  - machine independent and highly portable.
  - the server is not taken offline

Logical vs. Physical

- Physical
  - exact copies of database directories and files
  - faster than logical
  - more compact output
  - database level granularity (depend of the engine)
  - InnoDB table shares file storage with other InnoDB tables.)
  - can include any related files such as log or configuration files.
  - portable only to other machines that have identical or similar hardware
  - can be performed while the MySQL server is not running. If the server is running, you need to flush buffers and lock the tables

Logical Backup - Exercises

- Use MySQL Admin GUI tool
- Dump and restore with mysqldump and mysql
- Use SELECT ... INTO OUTFILE statement and LOAD DATA INFILE statement or the mysqlimport client

MySQL useful dump options

- `mysqldump [options] db_name [tables] > tabledump.sql`
- `mysqldump [options] --databases db_name1 [db_name2 db_name3...]   > databasesdump.sql`
- `mysqldump [options] --all-databases  > alldbsdump.sql`

- `--opt`
- `—compact`

Restoring a backup

- `source backup.sql`
- `cat backup.sql | mysql`
- `mysql < backup.sql`
- `copy/paste`

Exercise

- Dump and restore a database using both options (--opt and --compact)
- (Linux only) Compare the time. Use time command to measure the results.
- Using mysqldump create a full copy of NP database to NewNP.
- Using mysqldump create a copy of the emp table and call it old_emp.

hint

- mysqldump world Country | sed s/"Country"/"CopyOfCountry"/ | mysql world

Physical Backup

- cp, scp, tar, rsync
- mysqlhotcopy (cp for restore) for MyISAM (linux only)
- ibbackup (ibback for restore) for InnoDB (commercial version only)
- START BACKUP (ndb_restore) for NDB tables (outside the scope of this course)

Exercise

- Backup and restore the database with mysqlhotcopy and cp

Online versus offline backups

- Online
  - Less intrusive to other clients
  - Care must be taken to impose appropriate locking so that data modifications do not take place that compromise backup integrity
- Offline
  - Affects clients adversely because the server is unavailable during backup
  - Simpler backup procedure because there is no possibility of interference from client activity

Local versus remote backups

- Local
  - mysqldump can connect to local or remote servers.
  - SQL output (CREATE and INSERT statements), local or remote dumps can be done
  - delimited-text output (with the --taboption), data files are created on the server host
  - mysqlhotcopy performs only local backups
- Remote
  - SELECT ... INTO OUTFILE can be initiated from a remote client host, but the output file is created on the server host
  - Physical backup methods typically are initiated locally on the MySQL server host so that the server can be taken offline, although the destination for file copies might be remote.

Fast Remote Backup Example

- It is possible speed the process (from 6 hours to 42 seconds) up using compression and ssh tunneling
- `ssh production.nobleprog.net "mysqldump my_db | bzip2 -1" |  bzip2 -d > my_db_backup.tgz`

Full versus incremental backups

- An incremental backup consists of the changes made to the data since the full backup
- Incremental backups are made possible by enabling the server's binary log, which the server uses to record data changes
- One-way replication is an example of almost real-time backup

MySQL Log Files

Logs

- Log Output Destinations
- Error Log
- General Query Log
- Update Log
- Binary Log
- Slow Query Log
- MySQL Log Files

Log Output Destination and other options

- Turn logging on/of
- general_log=[{ON,OFF}]
- slow_query_log =[{ON,OFF}]
- Specify the path and filename
- general_log_file and slow_query_log_file
- if path is relative is starts in data directory
- Choosing Loging Format
- log-output=[TABLE,FILE,NONE]
- logging to tables incurs significantly more server overhead
- The session sql_log_off variable can be set to ON or OFF to disable or enable general query logging for the current connection.

The General Query Log

- Logs when clients connect or disconnect and each SQL statement received from clients
- Statements are written in the order that mysqld receives them, which might differ from the order in which they are executed.
- It contrasts to the binary log, for which statements are written after they are executed but before any locks are released
- You can disable the general query log at runtime:
  - SET GLOBAL general_log = ‘OFF'
  - SET GLOBAL general_log = 'ON'
- The session sql_log_off variable can be set to ON or OFF to disable or enable general query logging for the current connection

The Error Log

- The error log contains
  - information when mysqld was started and stopped
  - critical errors that occur while the server is running.
  - tables which need to be automatically checked or repaired
- You can specify the filename--log-error[=file_name]
- If no file_name value is given, mysqld uses the name host_name.err in the data directory
- The log_warnings system variable can be used to control warning logging to the error log (Enabled by default)
- If the value is greater than 1, aborted connections are written to the error log
- Mysqld_safe script can send the messages to syslog (Unix-like)

Slow Query Log

- Consists of all SQL statements:
- that took more than long_query_time seconds to execute and
- required at least min_examined_row_limit rows to be examined
- that do not use indexes are logged in the slow query log if the --log-queries-not-using-indexes option is specified
- the --log-slow-admin-statements server option enables you to request logging of slow administrative statements such as OPTIMIZE TABLE, ANALYZE TABLE, and ALTER TABLE to the slow query log

Slow Query Log

- For runtime control the global slow_query_log and slow_query_log_file system variables can be used
- You can process a slow query log file using the mysqldumpslow command to summarize the queries that appear in the log.

Test your setup using

- `select benchmark(100000000,1+1);`

Server Log Maintenance

- mysql-log-rotate (RedHat) automate log maintance
- In Debian/Ubuntu mysql-server package installs log rotate script (analyze /etc/logrotate.d/mysql-server script)
- For the binary log, you can set the expire_logs_days system variable to expire binary log files automatically after a given number of days
- You can force MySQL to start using new log files by flushing the logs. (FLUSH LOGS, mysqladmin flush-logs, mysqladmin refresh,mysqldump --flush-logs, mysqldump --master-data command)
- The binary log is flushed when its size reaches the value of the max_binlog_size

Server Log Maintenance

- A log flushing operation does the following:
  - General query and slow query - the server closes and reopens log files
  - Binary logging - the server closes the current binary log file and opens a new log file with the next sequence number
  - Error log - renames the error log with the suffix -old and creates a new empty error log file.
- To cause new general query and slow query log files to be created on Unix, rename the current logs before flushing them
  - mv mysql.log mysql.old
  - mysqladmin flush-logs
- You can disable the general query log or slow query log at runtime:
  - SET GLOBAL general_log = 'OFF';
  - SET GLOBAL slow_query_log = 'OFF';

What is Binary Log?

- contains all statements that update data (or possible could update the data)
- stored in the form of “events” that describe the modifications
- logs how long each statement took

Binary Log Purposes

- Replication
  - The binary log is used on master replication servers as a record of the statements to be sent to slave servers. The master server sends the events contained in its binary log to its slaves, which execute those events to make the same data changes that were made on the master.
- Point-in-time recovery
  - After a backup file has been restored, the events in the binary log that were recorded after the backup was made are re-executed. These events bring databases up to date from the point of the backup.

Point-in-time recovery

- Recovering first from the backup files to restore the server to its state when the backup was made,
- Re-executing changes in subsequently written binary log files to redo data modifications up to the desired point in time
- Because the output of mysqlbinlog includes SET TIMESTAMP statements before each SQL statement recorded, the recovered data and related MySQL logs will reflect the original times at which the transactions were executed

PiT Recovery Steps and Tools part 1

- Turn on binary log
  - `[mysqld]`
  - `log-bin=on`
- Dump the database
  - `` mysqldump > backup`date +%Y%m%d_%H%M%S`.sql ``
- Modify rows in the emp (useful stuff)
  - `insert into emp "badrecrod";`
- "Accidently" Drop the emp table
- View mysqlbinlog utility to view the binary log
- To find the binary logs
  - `SHOW MASTER STATUS;`
- To see a listing of all binary log files, use this statement:
  - `SHOW MASTER LOGS;`

PiT Recovery Steps and Tools part 2

- Drop all databases and restore the backup
  - `mysql < backupfile`
- Select the statements you want to execute
  - `mysqlbinlog binlogfile --start-datetime="2019-04-17 9:30:00" --stop-datetime="2019-04-17 9:59:59" > file.sql`
- Analyse file.sql
- Execute the commands
  - `mysql < file.sql`

Restoring in a specific log position

- More precise method about which part of the log to recover
  - Especially if many transactions occurred around the same time as a damaging SQL statement
- To determine the position numbers run the mysqlbinlog command with approximate time

  -
    ```
    mysqlbinlog --start-datetime="2014-04-20 9:55:00"
    --stop-datetime="2014-04-20 10:05:00" /var/log/
    mysql/bin.123456 > /tmp/mysql_restore.sql
    ```
- Analyse the mysql_restore.sql file and glean out the positions you are interested in
- Execute the queries

  -
    ```
    mysqlbinlog --start-position=368315 --stop-
    position=369312 /var/log/mysql/bin.123456 | mysql
    ```

Maintenance and Crash Recovery

- You can use myisamchk to check, repair, or optimize database tables
- Backup your database before repairing or optimizing
- With myisamchk, you must make sure that the server does not use the tables at the same time so that there is no unwanted interaction between myisamchk and the server
- Statements below can be used directly or by means of the mysqlcheck client program. One advantage of these statements over myisamchk is that the server does all the work.
  - CHECK TABLE
  - REPAIR TABLE.
  - OPTIMIZE TABLE.
  - ANALYZE TABLE

myisamchk with external locking disabled

- If you run mysqld with external locking disabled (which is the default), you cannot reliably use myisamchkto check a table when mysqld is using the same table
- If you can be certain that no one will access the tables through mysqld while you run myisamchk, you only have to execute mysqladmin flush-tables before you start checking the tables
- If you cannot guarantee this, you must stop mysqld while you check the tables.
- If you run myisamchk to check tables that mysqld is updating at the same time, you may get a warning that a table is corrupt even when it is not.

myisamchk with external locking enabled

- You can use myisamchk to check tables at any time.
- In this case, if the server tries to update a table that myisamchk is using, the server will wait for myisamchk to finish before it continues.
- If you use myisamchk to repair or optimize tables, you must always ensure that the mysqld server is not using the table (this also applies if external locking is disabled). If you do not stop mysqld, you should at least do a mysqladmin flush-tables before you run myisamchk.
- Your tables may become corrupted if the server and myisamchk access the tables simultaneously.

MyISAM files

  - tbl_name.frm Definition (format) file
  - tbl_name.MYD Data file
  - tbl_name.MYI Index file
- Each of these three file types is subject to corruption in various ways, but problems occur most often in data files and index files
- myisamchk works by creating a copy of the .MYD data file row by row.
- It ends the repair stage by removing the old .MYD file and renaming the new file to the original file name.

MyISAM files

- --quick does not create a temporary .MYD file, but instead assumes that the .MYD file is correct and generates only a new index file without touching the .MYD file. This is safe, because myisamchk automatically detects whether the .MYD file is corrupt and aborts the repair if it is.
- --quick --quick option (twice --quick) does not abort on some errors (such as duplicate-key errors) but instead tries to resolve them by modifying the .MYD file.
- Normally the use of two --quick options is useful only if you have too little free disk space to perform a normal repair. In this case, you should at least make a backup of the table before running myisamchk.

Checking MyISAM Tables for Errors

- myisamchk stops after the first error it finds the -v (verbose) option which keeps going, up through a maximum of 20 errors
- myisamchk tbl_name
    - Finds 99.99% of all errors
    - cannot find corruption that involves only the data file (which is very unusual)
- myisamchk -m tbl_name
    - Finds 99.999% of all errors.
    - First checks all index entries for errors and then reads through all rows.
    - Calculates a checksum for all key values in the rows and verifies that the checksum matches the checksum for the keys in the index tree.
- myisamchk -e tbl_name
    - Complete and thorough check of all data (-e means "extended check").
    - Does a check-read of every key for each row to verify that they indeed point to the correct row.
    - May take a long time for a large table that has many indexes.
- myisamchk -e -i tbl_name
    - The same as previous plus prints additional statistical information

CHECK TABLE

- CHECK TABLE works for MyISAM, InnoDB, ARCHIVE and CSV tables
- Check views for problems, such as tables that are referenced in the view definition that no longer exist
- The FOR UPGRADE option checks whether the named tables are compatible with the current version of MySQL

How to Repair Tables

- You can use myisamchk or REPAIR TABLE (for all engines)
- Myisamchk command deals only with MyISAM tables
- Shutdown your server before repair

Symptoms of Corrupted Tables

- queries that abort unexpectedly
- tbl_name.frm is locked against change
- Can't find file tbl_name.MYI (Errcode: nnn)
- Unexpected end of file
- Record file is crashed
- Got error nnn from table handler

Find out more about Errors

```
shell> perror 126 127 132 134 135 136 141 144 145
MySQL error code 126 = Index file is crashed
MySQL error code 127 = Record-file is crashed
MySQL error code 132 = Old database file
MySQL error code 134 = Record was already deleted (or record file crashed)
MySQL error code 135 = No more room in record file
MySQL error code 136 = No more room in index file
MySQL error code 141 = Duplicate unique key or constraint on write or update
MySQL error code 144 = Table is crashed and last repair failed
MySQL error code 145 = Table was marked as crashed and should be repaired
```

Repairing MyISAM tables

Stage 1

- Checking your tables
    - myisamchk *.MYI or myisamchk -e *.MYI
    - If the mysqld server is stopped, use the --update-state to mark the table as “checked.”

Repairing MyISAM tables

Stage 2

- Easy safe repair
  1. First, try myisamchk -r -q tbl_name (quick recovery mode, which attempts to repair the index file without touching the data file)
  2. If it didn't help, use the following procedure:
     1. Make a backup of the data file before continuing.
     2. Use myisamchk -r tbl_name (-r means "recovery mode")
        1. This removes incorrect rows and deleted rows from the data file
        2. Reconstructs the index file
  3. If the preceding step fails, use myisamchk --safe-recover tbl_name. Safe recovery mode uses an old recovery method that handles a few cases that regular recovery mode does not (but is slower)

Optimizing Table

  - `shell> myisamchk -r tbl_name`
- coalesce fragmented rows
- eliminate wasted space that results from deleting or updating rows
- other options
  - `--analyze,`
  - `--sort-index,`
  - `--sort-records=index_num`
- OPTIMIZE TABLE SQL
  - table repair
  - key analysis
  - also sorts the index tree so that key lookups are faster
  - there is also no possibility of unwanted interaction between a utility and the server

Running Multiple MariaDB Servers

Reasons for running multiple mysqld servers on the same machine

- To test a new release while leaving your existing production setup undisturbed
- Or you might want to give different users access to different mysqld servers that they manage themselves (service providers, projects, etc...)
- Two application use resources in different times (because of time zones, etc...) and you want to make use of these resources.
- Have more power which isn't available through virtualization
- Use different time zone and language settings

What needs to be different

- Connection to the server
  - --port=port_num
  - --socket=path
  - --bind-address=address
- PID filename
  - --pid-file=file_name
- Log files
  - --general_log or --log[=file_name]
  - --log-bin[=file_name]
  - --slow_query_log or --log-slow-queries[=file_name]
  - --log-error[=file_name]

What needs to be different (…contd)

- Temporary Directory
  - --tmpdir=path
- Data Directory
  - --datadir=path
- Configuration File (optional)
  - --defaults-file
- MySQL Binaries (optional)

Running Multiple Instances in Linux

- To compile MySQL binaries with different TCP/IP ports and Unix socket files so that each one is listening on different network interfaces
    - Use The MySQL Instance Manager (mysqlmanager) - DEPRICATED!
    - Use mysql_multi script

The mysql_multi script

```
[mysqld_multi]
mysqld     = /usr/bin/mysqld_safe
mysqladmin = /usr/bin/mysqladmin
user       = multi_admin
password   = multipass

[mysqld2]
socket     = /tmp/mysql.sock2
port       = 3307
pid-file   = /var/run/mysqld/mysqld2.pid
datadir    = /var/lib/mysql2
user       = mysql
```

Installing Service on Windows

- cd c:\mariadb1\bin
- mysql_install_db.exe --datadir=c:\mariadb1\data --service=MariaDB1
- cd c:\mariadb2\bin
- mysql_install_db.exe --datadir=c:\mariadb2\data --service=MariaDB2

Edit c:/mariadb2/data/my.ini and change port and tmp file

Step By Step

- Copy the data directory to /var/lib/mysql2
- Check the permission to the deamon user (usually mysql)
- Paste the configuration from the previous slide to /etc/mysql/my.cnf
- Run mysqld_multi start 2
- Connect using mysql -u root --port 3307 -p
- Grant shutdown privilege to each instance

  -
    ```
    GRANT SHUTDOWN ON *.* TO
    'multi_admin'@'localhost' IDENTIFIED BY
    ‘multipass';
    ```

Controlling Instances

- `shell> mysqld_multi [options] {start|stop|report} [GNR[,GNR] ...]`
- `shell> mysqld_multi stop 8,10-13`

Managing Configuration Files

- With --no-defaults, no option files are read.
- With --defaults-file=file_name, only the named file is read.
- Otherwise, option files in the standard list of locations are read, including any file named by the --defaults-extra-file=file_name option, if one is given. (If the option is given multiple times, the last value is used.)

MySQL Query Cache

MySQL Query Cache

- The Concept of Query Cache
- Testing Query Cache with SELECT
- Configuring Query Cache
- Checking Query Cache Status and Maintenance

The Query Cache

- Stores the text of a SELECT statement together with the corresponding result that was sent to the client.
- If an identical statement is received later, the server retrieves the results from the query cache rather than parsing and executing the statement again.
- The query cache is shared among sessions, so a result set generated by one client can be sent in response to the same query issued by another client.
- Useful in an environment where you have tables that do not change very often and for which the server receives many identical queries.
- This is a typical situation for many Web servers that generate many dynamic pages based on database content.
- When tables are modified, any relevant entries in the query cache are flushed.

The Query Cache

- The overhead for having the query cache active is 13% (the worst case scenario).
- Searches for a single row in a single-row table are 238% faster with the query cache than without it (minimum speedup)
- To disable the query cache at server startup, set the query_cache_size system variable to 0
- With some query cache configurations or server workloads, you might actually see a performance decrease:
- A query mix consisting almost entirely of a fixed set of SELECT statements is much more likely to benefit from enabling the cache than a mix in which frequentINSERT statements cause continual invalidation of results in the cache.

The Query Cache

- In some cases, a workaround is to use the SQL_NO_CACHE option to prevent results from even entering the cache for SELECT statements that use frequently modified tables, for example:
  - SELECT * FROM EMP;
  - INSERT INTO EMP...
    - Would work faster if mysql would insert the cache and invalidate it
  - SELECT SQL_NO_CACHE * FROM EMP;
  - INSERT INTO EMP...

How the Query Cache Operates

- Queries are compared before parsing, the queries below are regarded as different by the query cache:
  - SELECT * FROM tbl_name
  - Select * from tbl_name
- The cache is not used for queries of the following types:
- Queries that are a subquery of an outer query
- Queries executed within the body of a stored function, trigger, or event
- If a query result is returned from query cache, the server increments the Qcache_hits status variable, not Com_select.
- If a table changes, all cached queries that use the table become invalid and are removed from the cache
- INSERT, UPDATE, DELETE, TRUNCATE, ALTER TABLE, DROP TABLE, or DROP DATABASE changes the data,
- even if the modification doesn't effect the rows stored in the cache

When a query is NOT cached

- Uses BENCHMARK(), CONNECTION_ID(), CONVERT_TZ(), CURDATE() , etc…
- Uses User Defined Function or Stored Funcion
- Refers to user variables or local stored program variables
- Refers to tables in the mysql or INFORMATION_SCHEMA system database
- SELECT ... LOCK IN SHARE MODE SELECT ... FOR UPDATE SELECT ... INTO OUTFILE ... SELECT ... INTO DUMPFILE ... SELECT * FROM ... WHERE autoincrement_col IS NULL The last form is not cached because it is used as the ODBC workaround for obtaining the last insert ID value
- Statements within transactions that use SERIALIZABLE isolation level also cannot be cached because they use LOCK IN SHARE MODE locking
- Uses TEMPORARY tables
- Does not use any tables
- Generates warnings
- The user has a column-level privilege for any of the involved tables

Query Cache Configuration

- SET SESSION query_cache_type = OFF
- The maximum size of individual query results that can be cached can be set in query_cache_limit system variable (default 1MB).
- Maximum size that can be specified for the query cache at run time with the SET statement can be limited with --maximum-query_cache_size=32M option (in configuration file or parameter).
- Query Cache Configuration
- The query_cache_size system variable
- If 0 (zero) the query cache is disabled
- Individual clients can control cache behavior for the SESSION query_cache_type variable
- SET SESSION query_cache_type = OFF;

Query Cache Modes

- If the query cache size is greater than 0, the
- query_cache_type=[{ON,OFF,DEMAND}]
- variable influences how it works
- OFF prevents caching or retrieval of cached results.
- ON allows caching except of those statements that begin with SELECT SQL_NO_CACHE.
- DEMAND causes caching of only those statements that begin with SELECT SQL_CACHE.

Query Cache Min Reserved Unit

- The query cache allocates blocks for storing data on demand, when one block is filled, a new block is allocated.
- memory allocation operation is costly (timewise),
- the query cache allocates blocks with a minimum size given by the query_cache_min_res_unit (default 4KB)
- The default value of query_cache_min_res_unit is 4KB.
- If you have a lot of queries with small results, the big block size may lead to memory fragmentation, which triggers pruning (delete) queries from the cache due to lack of memory
- The number of free blocks and queries removed due to pruning are given by the values of the Qcache_free_blocks and Qcache_lowmem_prunes status variables.
- If most of your queries have large results (Qcache_total_blocks and Qcache_queries_in_cache), you can increase performance by increasing query_cache_min_res_unit

Query Cache Maintenance

- FLUSH QUERY CACHE
  - defragments the query cache to better utilize its memory
  - does NOT remove any queries from the cache
- RESET QUERY CACHE
  - removes all query results from the query cache
  - FLUSH TABLES statement also does this
- SHOW STATUS LIKE 'Qcache%';