FullStack.Cafe - Kill Your Tech Interview

Q1: What are different *integer data types* in MySQL? How can you use *unsigned integer* in MySQL? ☆☆

Topics: MySQL

Answer:

MySQL has different INT data types with different storage requirements including:

- **TINYINT** (1 byte) ,
- **SMALLINT** (2 byte),
- **MEDIUMINT** (3 byte),
- **INT** (4 byte) and
- **BIGINT** (8 byte).

All the INT types can be used as signed or unsigned. You can write UNSIGNED after data type to tell if it is unsigned.

For example below command will create a *Student* table which can have mark as unsigned **SMALLINT** value.

```
CREATE TABLE `Student`(
`name` VARCHAR(25) NOT NULL,
`marks` SMALLINT UNSIGNED);
```

Q2: Explain `DEFAULT` constraint in MySQL ☆☆

Topics: MySQL

Answer:

DEFAULT constraint provides a default value to a column. In case a row is inserted into the table and no value is provided to the column, the default value is taken.

Consider a table Customer created using statement below.

```
CREATE TABLE `Customer`(
    `customerid` CHAR(5) NOT NULL,
    `name` VARCHAR(25) NOT NULL,
    `country` VARCHAR(25) NOT NULL,
    PRIMARY KEY(`customerid`));
```

If we run **INSERT** command as below, we will get error as country name is not provided.

```
INSERT INTO `Customer` (`customerid`, `name`) VALUES ('12345','William Jones');
```

To fix this problem you can add a DEFAULT constraint using command below. Default value USA will be taken on running **INSERT** command above.

```
ALTER TABLE Customer ALTER country SET DEFAULT 'USA';
```

Q3: Explain *foreign key constraint* in MySQL ☆☆

Topics: MySQL

Answer:

A **Foreign key constraint** ensures cross referencing of values between two tables. A column or group of columns in a child table references a column or group of columns in another table. **Foreign key constraint** is defined in child table telling about the table and column(s) which are being referred. Foreign key constraint ensures referential integrity. Foreign key columns need to be indexed in MySQL, so an index is automatically created on foreign key column if index is not created on the column while creating a table with foreign key constraint.

Q4: What is *self referencing* foreign key? Give an example. ☆☆

Topics: MySQL

Answer:

A foreign key which is stored in a table itself is called to be **self referencing foreign key**.

For example consider an *Employee* database table. It has *employee_id* as primary key as well as a *manager_id* which is *employee_id* of his manager. If we create a foreign key constraint, as a manager is also an employee, *manager_id* will reference to *empolyee_id* in the same table. The *Employee* table with self referencing foreign key *manager_id* can be created using below statement.

```
CREATE TABLE `Employee`(
`name` VARCHAR(25) NOT NULL,
`employee_id` CHAR(9) NOT NULL,
`manager_id` CHAR(9) NOT NULL,
`salary` decimal(10,2) NULL,
PRIMARY KEY(`employee_id`),
FOREIGN KEY (manager_id) REFERENCES employee(employee_id) ON DELETE CASCADE
);
```

Q5: What is *Primary Key Constraint* and *Unique Key Constraints*? ☆☆

Topics: MySQL

Answer:

Primary key is a column or a group of columns in table which uniquely identifies a row in database. A primary key column can not have `NULL` value. Values in primary key column or columns must be unique so that a row or entry can be uniquely identified using primary key.

Unique key constraint ensures that values entered in a column are unique. Although primary key values will be unique, you might want to ensure that some other column also has non-duplicate entries. For example if we have multiple users and want to ensure that no two users with same email id are added to the table, we can put **UNIQUE** key constraint on *email_id* column. `NULL` values are ignored in case of Unique key constraint which means `NULL` is allowed value for email id in this case.

Q6: How are `VARCHAR` and `CHAR` different. Talk about cases where you will use one over other. ☆☆

Topics: MySQL

Answer:

Both **CHAR** and **VARCHAR** data types store characters up to specified length.

1. **CHAR** stores characters of fixed length while **VARCHAR** can store characters of variable length.
2. Storage and retrieval of data is different in **CHAR** and **VARCHAR**.
3. **CHAR** internally takes fixed space, and if stored character length is small, it is padded by trailing space characters. **VARCHAR** has 1 or 2 byte prefix along with stored characters.
4. **CHAR** has slightly better performance.
5. **CHAR** has memory allocation equivalent to the maximum size specified while **VARCHAR** has variable length memory allocation.

Q7: Describe `BLOB` in MySQL. What is it used for? ☆☆

Topics: MySQL

Answer:

BLOB or Binary Large Object can be used to store binary data in MySQL. Sometimes binary data like images need to be stored in SQL databases. For example you might want to store user photos along with other user details in the database table. Binary data of the user photo can be saved as a **BLOB**. By using **BLOB**, we will not require separate storage for images. **BLOB** helps in removing complexity and providing portability in such cases.

Q8: What is an `AGGREGATE` function. Name few aggregate functions used in MySQL. ☆☆

Topics: MySQL

Answer:

Aggregate functions are functions which are used to get a single summary value like minimum, maximum or average of a group of values.

> **COUNT**, **SUM**, **AVG**, **MIN**, **MAX** are examples of MySQL aggregate functions.

COUNT function is used to count number of entries returned by **SELECT** query. Similarly **SUM**, **AVG**, **MIN** and **MAX** can be used to calculate sum, average, minimum and maximum of a set of values returned by database query.

Q9: What is the difference between `TRUNCATE` and `DELETE`? ☆☆

Topics: MySQL SQL

Answer:

- **DELETE** is a Data Manipulation Language(DML) command. It can be used for deleting some specified rows from a table. **DELETE** command can be used with **WHERE** clause.

- **TRUNCATE** is a Data Definition Language(DDL) command. It deletes all the records of a particular table. **TRUNCATE** command is faster in comparison to **DELETE**. While **DELETE** command can be rolled back, **TRUNCATE** can not be rolled back in MySQL.

Q10: What is the difference between *Data Definition Language (DDL)* and *Data Manipulation Language (DML)*? ☆☆

Topics: MySQL SQL T-SQL Databases

Answer:

- **Data definition language (DDL)** commands are the commands which are used to define the database. **CREATE**, **ALTER**, **DROP** and **TRUNCATE** are some common DDL commands.

- **Data manipulation language (DML)** commands are commands which are used for manipulation or modification of data. **INSERT**, **UPDATE** and **DELETE** are some common DML commands.

Q11: What is a `VIEW` in MySQL. How can you create and query a view? ☆☆

Topics: MySQL

Answer:

Views are **virtual tables** which are stored SQL queries with certain given name. **VIEW** replaces a **SELECT** query with a given named view. The **SELECT** query could be complex **SELECT** query across different database tables or a simple **SELECT** query. Once invoked, a view gives results using the query stored in view.

A view can be created using below syntax.

```
CREATE VIEW `name_of_view`
AS SELECT select_statement;
```

A query can be made on a view similar to database table like below.

```
SELECT * FROM  name_of_view;
```

Q12: What are *Derived Columns*. What possible problems can a derived column pose? ☆☆☆

Topics: MySQL

Answer:

A **derived column** is a column in database table whose_ value depends on one or more columns_.

For example consider a *Customer* table for a gym in which *date of birth, age, height, weight* and *BMI* of a customer are stored. As *age* of a person can be derived from his *date of birth*, *age* is a derived variable. Similarly as *BMI* can be derived from *height* and *weight* of the customer, *BMI* is a derived column.

The issue with **derived columns** is that it keeps *redundant* data. Redundant data might further affect data **accuracy** and **consistency**. For example if an update changes *date of birth* of a user but somehow fails to change his *age*, the data stored in table becomes inaccurate.

Q13: Explain the use of `FEDERATED` tables in MySQL ☆☆☆

Topics: MySQL

Answer:

FEDERATED tables are tables through which MySQL provides a way to access database tables located in remote database servers. Actual physical data resides in remote machine but the table can be accessed like a local table. To use a federated table **ENGINE=FEDERATED** and a connection string containing user, remote hostname, port, schema and table name are provided in **CREATE TABLE** command something like below.

```
CREATE TABLE table_fed (
 ...
)
ENGINE=FEDERATED
CONNECTION='mysql://user@remote_hostname:port/federated_schema/table';
```

Q14: Explain `GRANT` command in MySQL ☆☆☆

Topics: MySQL

Answer:

When a new MySQL user is created, he requires certain privileges to perform various database operations. **GRANT** command grants certain privileges to the user. For example below statement grants permission to run **SELECT** and **INSERT** on **TABLE** *customertable* to user *username@localhost*.

```
GRANT SELECT, INSERT ON customertable TO 'username'@'localhost'
```

Q15: How can `VIEW` be used to provide *security layer* for your app? ☆☆☆

Topics: MySQL SQL

Answer:

Views can be used to selectively *show limited data* to a user.

For example consider a *Customer* table which has columns including *name*, *location* and *credit card number*. As credit card number is a sensitive customer information, it might be required to hide credit card number from a certain database user. As the user can not be given access to entire *Customer* table, a view with *name* and *location* of the *Customer* can be created. The database user can be given access to that view only. This way view provides a security layer ensuring limited access to the database table.

Q16: Both `TIMESTAMP` and `DATETIME` are used to store data and time. Explain difference between them and when should one be used? ☆☆☆

Topics: MySQL

Answer:

Both **TIMESTAMP** and **DATETIME** store date time in *YYYY-MM-DD HH:MM:SS* format. While **DATETIME** stores provided date time, **TIMESTAMP** first converts provided time to UTC while storing and then again converts it back to server time zone upon retrieval. So if you need to serve different users in different countries using same time data, **TIMESTAMP** facilitates it. **DATETIME** simply stores provided date time without making any time zone related conversion.

Q17: What is `mysqldump`? ☆☆☆

Topics: MySQL

Answer:

mysqldump is a command line utility for **backup and restore** of database. It can be run easily to generate dump of stored databases using command line.

mysqldump is very simple to use and comes with MySQL installation. It generates statements to create tables as well as update data. As it does logical backup, backup and restore process can be very slow for large databases. Dump file generated by **mysqldump** is human readable. A simple command like below will dump database using **mysqldump** to a human readable file which can be further compressed or stored.

```
mysqldump --user root -p databasename > file.sql
```

Q18: What is the difference between commands `create database` and `create schema` in MySQL? ☆☆☆

Topics: MySQL

Answer:

Terms **database** and **schema** are synonymous in MySQL. Some other enterprise level databases like Oracle and Microsoft SQL server make distinction between database and schema. A MySQL developer can interchangeably use both the terms. For example a database called *test* can be created by using either of the CREATE statements below.

```
    CREATE DATABASE test;
    CREATE SCHEMA test;
```

Q19: What is *index* in MySQL? What is advantage of index? ☆☆☆

Topics: MySQL

Problem:

How can you create and use index in MySQL?

Solution:

While retrieving rows with certain column values like using **WHERE** clause, the database iterates through each entry to search for matching records. The retrieval can become significantly fast by creating index on those columns. Most of MySQL index are stored in *B Tree* with some exceptions. Index can be created in a database by using below syntax.

```
CREATE INDEX index_name ON table_name (`column_name`);
```

Q20: What is a *trigger*. What are different type of triggers in MySQL? ☆☆☆

Topics: MySQL

Answer:

Triggers are database objects or programs which are are *invoked on certain events automatically*. Triggers are invoked before or after insert, update or delete on rows of associated table.

There are six type of events associated with a database table when a trigger can be invoked. It includes:

- **BEFORE INSERT**,
- **AFTER INSERT**,
- **BEFORE UPDATE**,
- **AFTER UPDATE**,
- **BEFORE DELETE**
- **AFTER DELETE**.

For example **BEFORE UPDATE** trigger will automatically invoked and run before update is called on the table.

FullStack.Cafe - Kill Your Tech Interview

Q1: What different *stored objects* are supported in MySQL? ☆☆☆

Topics: MySQL

Answer:

Different stored objects in MySQL include **VIEW, STORED PROCEDURE, STORED FUNCTION, TRIGGER, EVENT**.

- **VIEW** - It is a virtual table based on a result set of a database query.
- **STORED PROCEDURE** - It is a procedure stored in database which can be called using CALL statement. Stored procedure does not return a value.
- **STORED FUNCTION** - It is like function calls which can contain logic. It returns a single value and can be called from another statement.
- **TRIGGER** - Trigger is program which is associated with a database table which can be invoked before or after insert, delete or update operations.
- **EVENT** - Event is used to run a program or set of commands at defined schedule.

Q2: What is the use of `IN` and `BETWEEN` in MySQL queries? ☆☆☆

Topics: MySQL

Answer:

BETWEEN operator can be used to select value in certain range. For example below statement will return list of students with *rollnumber* between 12 and 17.

```
SELECT * FROM `Student` WHERE `rollnumber` BETWEEN 12 AND 17;
```

IN operator is used to select rows which have a column value in provided list of values. Below command lists all students with *rollnumber* 2, 4 and 7.

```
SELECT * FROM `student` WHERE `rollnumber` IN ('2', '4', '7');
```

Q3: What is *Stored Function* in MySQL. How are they different from *Stored Procedure*? ☆☆☆

Topics: MySQL

Answer:

Stored function is a stored complex logic which can be executed like a function call from any other statement. It returns a single value. It can be used to store business logic and formulas in database. Stored functions can even run **SELECT** command or table manipulation commands like **INSERT** and **UPDATE**.

The most general difference between procedures and functions is that they are invoked differently and for different purposes:

1. A procedure does not return a value. Instead, it is invoked with a CALL statement to perform an operation such as modifying a table or processing retrieved records.
2. A function is invoked within an expression and returns a single value directly to the caller to be used in the expression.
3. You cannot invoke a function with a CALL statement, nor can you invoke a procedure in an expression.

e.g. `SELECT get_foo(myColumn) FROM mytable` is not valid if `get_foo()` is a procedure, but you can do that if `get_foo()` is a function. The price is that functions have more limitations than a procedure.

```
CREATE PROCEDURE proc_name ([parameters])
 [characteristics]
 routine_body


CREATE FUNCTION func_name ([parameters])
 RETURNS data_type       // diffrent
 [characteristics]
 routine_body
```

Q4: What are `REPEAT`, `LOOP` and `WHILE` statements used for? ☆☆☆

Topics: MySQL

Answer:

REPEAT, **LOOP** and **WHILE** are flow control statements which can be used in MySQL stored functions and stored procedures. They have slightly different syntax.

- **LOOP** creates a simple loop and *LEAVE* statement can be used to come out of LOOP.
- **WHILE** can be used as *WHILE condition_is_met DO* which exits when condition is false.
- **REPEAT** repeats the loop *UNTIL* condition is met.

Q5: What is the use of `DELIMETER` command in MySQL? ☆☆☆

Topics: MySQL

Answer:

MySQL workbench or MySQL client use **;** as delimiter to separate different statements. **DELIMITER** command can be used to change delimiter in MySQL from **;** to something else. It is used while writing trigger and stored procedures in MySQL. Command below make **//** as delimiter.

```
DELIMITER //
```

For example in a stored procedure, **";"** is part of the actual stored procedure and not a delimiter. So while writing a stored procedure, we can make something else like // as delimiter and afterward revert it back by calling below command.

```
DELIMITER ;
```

Q6: What are *key constraints*. What different types of constraints are there in MySQL? ☆☆☆

Topics: MySQL

Answer:

Key constraints are rules which ensure that correct data is stored in the database. It ensures integrity of the data. **MySQL** constraints include:

- **PRIMARY KEY CONSTRAINT** - creates index using primary key column or group of columns for faster access to database table. Primary key can not have null or duplicate values.
- **FOREIGN KEY CONSTRAINT** - creates link between columns of two different tables, ensures that a particular value assigned to a column in one table has a matching entry in another table.
- **NOT NULL CONSTRAINT** - ensures a particular column cannot have null values.
- **UNIQUE CONSTRAINT** - ensures a particular column does not contain duplicate values. Unlike primary key constraint, there can be more than one columns with unique key constraint in a table.
- **CHECK CONSTRAINT** - can be used to ensure a certain condition is met while assigning a value to a column.
- **DEFAULT CONSTRAINT** - assigns default value to a column if a value is not provided.

Q7: What is difference between `BLOB` and `TEXT` in MySQL? ☆☆☆

Topics: MySQL

Answer:

- **BLOB** data types are designed to store binary data like picture or video in database.
- **TEXT** data types are designed to store large text data.

BLOB stores binary byte string while **TEXT** stores character string. Although **BLOB** can be used for storing text data, **TEXT** data types support sorting and comparison around text which is not supported by **BLOB**.

There are four **TEXT** data types including **TINYTEXT**, **TEXT**, **MEDIUMTEXT** and **LONGTEXT** which can hold up to 255 characters, 65,535 characters, 16,777,215 characters and 4,294,967,295 characters respectively. Similarly four related **BLOB** types including **TINYBLOB**, **BLOB**, **MEDIUMBLOB** and **LONGBLOB** can hold up to 255 bytes, 65,535 bytes, 16,777,215 bytes and 4,294,967,295 bytes respectively.

Q8: What happens if a parent row which is referenced by child row is being *deleted* in case of *foreign key* constraint? ☆☆☆

Topics: MySQL

Answer:

The behavior of what happens depends upon set referential actions including **CASCADE**, **SET NULL**, **RESTRICT**, **NO ACTION** which can be provided while creating a foreign key constraint using **ON UPDATE** and **ON DELETE** commands. If no such value is provided while creating foreign key constraint, **RESTRICT** will be default action.

- **CASCADE** - changes will be cascaded which means rows in child table referencing parent table will be deleted.
- **SET NULL** - value in child table will be set to null.
- **RESTRICT** - deletion of parent row will fail.
- **NO ACTION** - it works same as RESTRICT in MySQL.

Q9: What are different `TEXT` data types in MySQL. What is difference between `TEXT` and `VARCHAR`? ☆☆☆

Topics: MySQL

Answer:

Different text data types in **MySQL** include:

- **TINYTEXT**,
- **TEXT**,
- **MEDIUMTEXT** and
- **LONGTEXT**.

These data types have different maximum size. While **TINYTEXT** can hold string up to 255 characters, **TEXT** can hold up to 65,535 characters, **MEDIUMTEXT** can hold up to 16,777,215 characters and **LONGTEXT** can hold up to 4,294,967,295 characters.

VARCHAR is also a variable text data type with some difference. **VARCHAR** is stored inline in the database table while **TEXT** data types are stored elsewhere in storage with its pointer stored in the table. A prefix length is must for creating index on **TEXT** data types. **TEXT** columns do not support default values unlike **VARCHAR**.

Q10: What is MySQL *Workbench*? ☆☆☆

Topics: MySQL

Answer:

MySQL Workbench is a visual tool which database architects, developers and administrators can use. It can be used for data modeling, database development, database administration, improving MySQL performance as well as database migration.

- **Data Modeling** - MySQL workbench provides data modeling features like visual tool for ER modeling, reverse and forward engineering to and from SQL scripts.
- **MySQL development** - Provides SQL editor with features like auto-completion, syntax highlighting. Object browser is also there.
- **Database administration** - Provides visual tools for database administration tasks like database instance start/stop, server configuration, user administration.
- **Database migration** - Provides migration solution and supports popular databases like Microsoft Access, Microsoft SQL server, Sybase ASE, SQLite and PostgresSQL. Also helps in migration from older MySQL versions.

Q11: What are some major differences between MySQL and Oracle database? ☆☆☆☆

Topics: MySQL

Answer:

Both **Oracle** and **MySQL** are relational database management systems supporting SQL. Here are some major differences between **Oracle** and **MySQL**.

- **MySQL** is open source and free. **Oracle** provides express edition for free but it comes with limited features. **Oracle** provides enterprise level tools.

- **Oracle** is better in terms of scalability. **Oracle** is suited for large scale enterprise level projects while **MySQL** is suited for small and medium scale projects.
- While **MySQL** is Relational DBMS, **Oracle** has features of Object Relational DBMS. It supports more complex object relational model in additional to Relational DBMS supported by **MySQL**.
- **MySQL** uses `mysqldump` or `mysqlhotcopy` for data backup. **Oracle** supports many options including Export and Import, Offline backup, Online backup, using RMAN utility, Export or Import Data Pump.
- PL/SQL is supported in **Oracle** while **MySQL** does not support PL/SQL.
- **MySQL** uses username, password and host for security. Apart from username, password **Oracle** database provides features like profile validation for security.

Overall **MySQL** is good for small and medium sized websites and read only websites. **Oracle** is good for highly scalable enterprise level applications.

Q12: Which *partitioning types* does MySQL support? ☆☆☆☆

Topics: MySQL

Answer:

MySQL supports range partitioning, list partitioning, hash partitioning and key partitioning.

- **Range partitioning** - rows are partitioned based on a certain range of column value.
- **List partitioning** - rows are partitioned based on if column value is same as value provided in a list.
- **Hash partitioning** - rows are partitioned based on hash of column value. Hash partitioning provides a more even partitioning. Hash is computed based on user defined hash function.
- **Key partitioning** - rows are partitioned based on a hashing function provided by MySQL like MD5 hash.

Q13: What does `OPTIMIZE TABLE` command do in MySQL? ☆☆☆☆

Topics: MySQL

Answer:

A database table can be defragmented over the time. `OPTIMIZE TABLE` command can be executed to reorganize table data and index. `OPTIMIZE TABLE` command might be useful for tables which are **frequently updated**. It can help improve performance of I/O operations.

Q14: What is *database engine* or *storage engine*? Mention few storage engines supported by MySQL and their use. ☆☆☆☆

Topics: MySQL

Answer:

Database engines or storage engines are software programs which perform database operations like create, read, update and delete. Major storage engines supported by MySQL are **InnoDB** and **MyISAM**. Other storage engines supported by MySQL include **MEMORY, CSV, FEDERATED, MERGE, ARCHIVE** and **BLACKHOLE** storage engines.

- **InnoDB** storage engine - default MySQL storage engine from version 5.5 and later. InnoDB is ACID compliant, provides transaction support and foreign key support.
- **MyISAM** storage engine- default storage engine before 5.5. It is non-transactional.
- **MEMORY** storage engine- it creates temporary database tables in memory.

- **CSV** storage engine - it uses csv file for storing data.
- **FEDERATED** storage engine - helps access data physically located in remote database server using a query to local database server.
- **MERGE** storage engine - can be used to merge more than one table with identical column data into one table.
- **ARCHIVE** storage engine - uses *zlib* compression to store archived data using less space, does not support indexing.
- **BLACKHOLE** storage engine - acts like a black hole, accepts data to store, does not store it and always returns empty. BLACKHOLE storage engine can be used in performance testing.

Q15: What are differences between *MyISAM* and *InnoDB* database engines commonly used in MySQL? ☆☆☆☆

Topics: MySQL

Answer:

Row level locking, foreign key support and transacation support are main features which differentiate InnoDB from MyISAM.

- **MyISAM** was default storage engine before 5.5 while **InnoDB** is default storage engine from 5.5 and later versions.
- **InnoDB** is ACID compliant ensuring data integrity while **MyISAM** is not ACID compliant.
- **COMMIT** and **ROLLBACK** are supported by InnoDB.
- **InnoDB** has row level locking, which makes it faster in highly concurrent cases in comparison to **MyISAM** which has locking at table level. With row level locking multiple sessions can simultaneously write in a table.
- **Foreign key constraint** is supported only in **InnoDB** and not **MyISAM**. InnoDB is appropriate engine for payment related applications where transaction support becomes important.

Q16: Compare MySQL and PostgresSQL ☆☆☆☆

Topics: MySQL

Answer:

- **MySQL** is a **RDBMS** (relational database) while **PostgresSQL** is an **ORDBMS** (object relational database) which means apart from relational database, it also supports some object oriented language features like table inheritance and functional overloading.

- Both **MySQL** and **PostgresSQL** are open source and free.

- **MySQL** has better performance in case of simple queries. It gets this advantage as MySQL has less features in comparison to **PostgresSQL**. **PostgresSQL** has better support for complex queries.

- **MySQL** is more popular than **PostgresSQL** so developer can get better support.

- **PostgresSQL** is fully ACID compliant while **MySQL** is ACID compliant while using InnoDB. **PostgresSQL** is better choice than mysql for highly reliable applications.

- Both **PostgresSQL** and **MySQL** have some **NoSQL** support but **PostgresSQL** has better support to NoSQL. For example **PostgresSQL** has better support to Json, XML and geometric types in comparison to MySQL.

- **PostgresSQL** is compliant to SQL standard while **MySQL** is compliant to SQL standards to lesser extent.

Q17: What is advantage of `FULLTEXT` over `LIKE` for performing text search in MySQL? ☆☆☆☆

Topics: MySQL

Answer:

LIKE is used in MySQL for simple partial text matching. **FULLTEXT** can be used for performing search like search engines. It uses data indexing which means the searches are faster unlike **LIKE** where entire table needs to be scanned. Search index of table data is updated dynamically on its own. **FULLTEXT** performs flexible searching and natural language text search is supported as well. **FULLTEXT** also provides a relevance ranking of results.

Q18: What is *master-slave replication* in MySQL? What are its advantages? ☆☆☆☆

Topics: MySQL

Answer:

Master slave replication helps keep a live copy of the data from a master database in another server. All write operations are done on master server. The updates are further replicated in slave server.

- Master slave replication can provide scalability as read operations can easily be distributed among different servers thus providing load balancing.
- As a slave server keeps live copy of data. A slave can quickly be made master in case master server is down, thus it helps reduce downtime.
- Data stored in slave can act as data backup as well.
- Different experiments and analytics can be run on slave server without putting additional load on master server.

Q19: What is cursor used in MySQL? What are properties of MySQL *cursor*? ☆☆☆☆

Topics: MySQL

Answer:

Cursor is used in stored programs like stored procedure and stored functions. **Cursor** is a type of loop which can iterate through result of a **SELECT** query inside stored programs. Here are some properties of cursor in MySQL.

- **Read-only** - In MySQL, cursor is read only which means data of table cannot be updated using cursor.
- **Non-scrollable** - It is non-scrollable which means one can traverse in forward direction with one entry at a time without skipping entries or moving in reverse direction.
- **Asensitive** - It is asensitive which means cursor will not make temporary copy of the data rather use original data. One issue with cursor being asensitive is that if some other process changes the data while cursor is using it, result might not be predictable.

Consider:

```
CREATE PROCEDURE curdemo()
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE a CHAR(16);
  DECLARE b, c INT;
```

```
  DECLARE cur1 CURSOR FOR SELECT id,data FROM test.t1;
  DECLARE cur2 CURSOR FOR SELECT i FROM test.t2;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

  OPEN cur1;
  OPEN cur2;

  read_loop: LOOP
    FETCH cur1 INTO a, b;
    FETCH cur2 INTO c;
    IF done THEN
      LEAVE read_loop;
    END IF;
    IF b < c THEN
      INSERT INTO test.t3 VALUES (a,b);
    ELSE
      INSERT INTO test.t3 VALUES (a,c);
    END IF;
  END LOOP;

  CLOSE cur1;
  CLOSE cur2;
END;
```

Q20: What are some advantages and disadvantages of *stored procedures* in MySQL? ☆☆☆☆

Topics: MySQL

Answer:

Here are some **advantages** of **stored procedure**:

- **Stored procedures** help in providing security to database as users can be provided access just to required stored procedures for limited operations defined in the stored procedure instead of giving access to tables. It also helps in logging.

- **Stored procedures** can be reused across different applications using the database thus provide reusability. Business logic resides in database via stored procedures. Thus even if different applications are written in different languages, they share same logic via stored procedures.

- As a large query can be saved using stored procedure, it helps reduce network traffic as there is no need of repeated calls to same large query, instead the database request will just make a call to stored procedure along with required arguments.

Here are some **disadvantages**:

- Stored procedures can not code complex business logic.
- Also stored procedures are difficult to debug and maintain.
- Running stored procedures in database server means more workload on database server.

FullStack.Cafe - Kill Your Tech Interview

Q1: How can you *delete one or multiple columns* from a MySQL database table? ☆☆

Topics: MySQL

Answer:

DROP COLUMN command can be used to drop a column from MySQL table. Below statement can be used to drop a column.

```
ALTER TABLE table_name
DROP COLUMN column_to_delete;
```

Similarly below syntax can be used to drop multiple columns.

```
ALTER TABLE table_name
DROP COLUMN column_to_delete1,
DROP COLUMN column_to_delete2;
```

Q2: Find *duplicate* values in a SQL table ☆☆☆

Topics: SQL MySQL

Problem:

We have a table

```
ID   NAME   EMAIL
1    John   asd@asd.com
2    Sam    asd@asd.com
3    Tom    asd@asd.com
4    Bob    bob@asd.com
5    Tom    asd@asd.com
```

I want is to get duplicates with the same `email` and `name`.

Solution:

Simply group on both of the columns:

```
SELECT
    name, email, COUNT(*) as CountOf
FROM
    users
GROUP BY
    name, email
HAVING
    COUNT(*) > 1
```

Q3: How can `ENUM` be used in MySQL. Give an example. ☆☆☆

Topics: MySQL

Answer:

`ENUM` can be used to set a column as **enum** type. `ENUM` in MySQL is string object which can take one of the permitted value. In example below, *country* column can have one of the three values provided:

```
CREATE TABLE `Student`(
`rollnumber` INT NOT NULL,
`name` VARCHAR(25) NOT NULL,
`country` ENUM('USA', 'UK', 'Australia'),
PRIMARY KEY(`rollnumber`));
```

Consider:

```
INSERT INTO `Student` values('6', 'John', 'USA');
```

Q4: How can you *add one or multiple columns* after a certain column in an existing MySQL table? ☆☆☆

Topics: MySQL

Answer:

ADD COLUMN command can be used to add a column to existing database table. For example below syntax can be used to add one or more ***VARCHAR*** columns after *existingColumn*.

```
ALTER TABLE tableName
ADD COLUMN newColumn1 VARCHAR(25) AFTER existingColumn,
ADD COLUMN newColumn2 VARCHAR(25) AFTER newColumn1;
```

Q5: What is `AUTO INCREMENT` in MySQL? Explain with an example. ☆☆☆

Topics: MySQL

Answer:

- **AUTO INCREMENT** in MySQL is used to automatically assign next unique integer value to a particular column.
- **AUTO INCREMENT** can be used to generate unique id for each inserted row without assigning a value to it. In MySQL, only columns which keep unique values like column with **UNIQUE CONSTRAINT** or **PRIMARY KEY** can be marked for **AUTO INCREMENT**. A table can have only one column marked for **AUTO INCREMENT**.

Code below can be used to mark *studentid* in *Student* table to auto increment. On adding a new *Student* without providing *studentid*, a unique student id with next available value is generated and assigned to the row.

```
CREATE TABLE `student`(`studentid` INT NOT NULL AUTO_INCREMENT,  `name` VARCHAR(25) NOT NULL, PRIMARY
KEY(`studentid`));
```

Q6: What happens if a *duplicate entry* is already there when adding constraint? ☆☆☆

Topics: MySQL

Problem:

You are given a table called **Customer** with attributes including name and email. How can you add a constraint to the table such that no two customers with same name and email can be added. What happens if a duplicate entry is already there?

Solution:

You can use **ALTER TABLE** command to add a **UNIQUE** key constraint on *name* and *email* columns. Below statement will do the job.

```
ALTER TABLE `Customer` ADD UNIQUE (`name`, `email`, );
```

The statement will fail if two entries with same name and email are already existing the the table.

Q7: A multiple column index is created over `firstName`, `lastName`, `city` columns of a `Customer` table. Will this index be used for `SELECT` queries based on only `first_name`, only `last_name` or only `city` values? ☆☆☆☆

Topics: MySQL

Answer:

Multiple column index works on **left most prefix of the indexed columns**. It means the index will work on queries based on *first_name* only and index will not be used in queries based on *last_name*. To elaborate the point following queries will use the index.

```
SELECT * FROM CUSTOMER WHERE firstName='Smith';
SELECT * FROM CUSTOMER WHERE firstName='Smith' AND lastName='Jones';
SELECT * FROM CUSTOMER WHERE firstName='Smith' AND lastName='Jones' AND city='London';
```

The queries like below will not use the index.

```
SELECT * FROM CUSTOMER WHERE lastName='Jones' AND city='London';
SELECT * FROM CUSTOMER WHERE lastName='Jones';
SELECT * FROM CUSTOMER WHERE city='London';
```

Q8: Write a query to *concatenate* and display all `DISTINCT` names in a single row ☆☆☆☆

Topics: MySQL

Problem:

Given a table **Student** with a column *firstname*. More than one student can have same first name. Write a query to concatenate and display all distinct names in a single row.

Solution:

GROUP_CONCAT concatenates multiple values in a group. It can be used to get concatenated names in a single row using below command:

```
SELECT GROUP_CONCAT(DISTINCT name) FROM Student;
```

Q9: Provide an example of `UPSERT` logic using MySQL ☆☆☆☆

Topics: MySQL

Problem:

Write MySQL command to update a table with certain values but with an extra condition: if the row to update is not found you need to create a new entry.

Solution:

Use **ON DUPLICATE KEY UPDATE** to run a command which can do an **INSERT** and **UPDATE** if needed in a single statement.

For example if we have a **User** table with attributes *userid*, *name* and *mobilenumber*. Following statement will try to insert Alice to the *User* table. If a user with provided *userid* is already present in the table, the command will simply update the row.

```
INSERT INTO `User` (`userid`, `name`, `mobilenumber`)
VALUES('11112227', 'Alice', '876876876')
ON DUPLICATE KEY UPDATE name='Alice', mobilenumber='876876876';
```

Q10: What is `autocommit` in MySQL? Can you run a transaction *without* disabling autocommit? ☆☆☆☆

Topics: MySQL

Answer:

In MySQL, if *autocommit* is disabled, any command you run will not be committed automatically, which means changes made through these commands will become permanent only if **COMMIT** is called. By default *autocommit* is enabled in MySQL, which means any change we make is part of a single transaction, *commit* is done automatically and it can not be rolled back.

If *autocommit* is enabled you can call **START TRANSACTION** to start a transaction. **START TRANSACTION** command makes the changes temporary until either **COMMIT** or **ROLLBACK** is called to end the transaction.

Q11: What is *Memory Storage Engine* in MySQL? What are heap tables? ☆☆☆☆

Topics: MySQL

Answer:

In MySQL, **Memory storage engine** is used to store temporary data in memory. As data is stored in memory, memory storage engine can be used for high performance non-critical data. Memory storage engine can be used for data which is updated very frequently. It can be used for caching. Memory storage engine was called **heap table** in earlier versions of MySQL, so sometimes both these terms are used interchangeably.

If server restarts, the temporary data stored in Memory storage engine is lost. Some variable length data types like BLOB and TEXT data types are not supported by memory storage engine although VARCHAR is supported.

Q12: What is faster, *one big* query or *many small* queries? ☆☆☆☆

Topics: MySQL SQL

Answer:

What would address your question is the subject JOIN DECOMPOSITION.

You can decompose a join by running multiple single-table queries instead of a multitable join, and then performing the join in the application. For example, instead of this single query:

```
SELECT * FROM tag
JOIN tag_post ON tag_post.tag_id = tag.id
JOIN post ON tag_post.post_id = post.id
WHERE tag.tag = 'mysql';
```

You might run these queries:

```
SELECT * FROM tag WHERE tag = 'mysql';
SELECT * FROM tag_post WHERE tag_id=1234;
SELECT * FROM post WHERE post.id IN (123,456,567,9098,8904);
```

Why on earth would you do this? It looks wasteful at first glance, because you've increased the number of queries without getting anything in return. However, such restructuring can actually give significant performance advantages:

- Caching can be more efficient. Many applications cache "objects" that map directly to tables. In this example, if the object with the tag `mysql` is already cached, the application will skip the first query. If you find posts with an ID of 123, 567, or 908 in the cache, you can remove them from the `IN()` list. The query cache might also benefit from this strategy. If only one of the tables changes frequently, decomposing a join can reduce the number of cache invalidations.
- Executing the queries individually can sometimes reduce lock contention
- Doing joins in the application makes it easier to scale the database by placing tables on different servers.
- The queries themselves can be more efficient. In this example, using an `IN()` list instead of a join lets MySQL sort row IDs and retrieve rows more optimally than might be possible with a join.

- You can reduce redundant row accesses. Doing a join in the application means retrieving each row only once., whereas a join in the query is essentially a denormalization that might repeatedly access the same data. For the same reason, such restructuring might also reduce the total network traffic and memory usage.
- To some extent, you can view this technique as manually implementing a hash join instead of the nested loops algorithm MySQL uses to execute a join. A hash join might be more efficient.

As a result, doings joins in the application can be more efficient when you cache and reuse a lot of data from earlier queries, you distribute data across multiple servers, you replace joins with `IN()` lists, or a join refers to the same table multiple times.

Q13: What are the differences between MongoDB and MySQL?

☆☆☆☆☆

Topics: MongoDB MySQL

Answer:

The Major Differences between MongoDB and MySQL are:

1. There is a difference in the representation of data in the two databases. In MongoDB, data represents in a collection of JSON documents while in MySQL, data is in tables and rows. JSON documents can compare to associative arrays when using PHP and directory objects when using Python.
2. When it comes to querying, you have to put a string in the query language that the DB system parses. The query language is called Structured Query Language, or SQL,from where MySQL gets its name. This exposes your DB susceptible to SQL injectionattacks. On the other hand, MongoDB's querying is object-oriented, which means you pass MongoDB a document explaining what you are querying. There is no parsing whatsoever, which will take some time getting used to if you already use SQL.
3. One of the greatest benefits of relational databases like MySQL is the JOIN operation. The operation allows for the querying across several tables. Although MongoDB doesn't support joints, it supports multi-dimensional data types like other documents and arrays.
4. With MySQL, you can have one document inside another (embedding). You would have to create one table for comments and another for posts if you are using MySQL to create a blog. In MongoDB, you will only have one array of comments and one collection of posts within a post.
5. MySQL supports atomic transactions. You can have several operations within a transaction and you can roll back as if you have a single operation. There is no support for transactions in MongoDB and the single operation is atomic.
6. One of the best things about MongoDB is that you are not responsible for defining the schema. All you need to do is drop in documents. Any 2 documents in a collection need not be in the same field. You have to define the tables and columns before storage in MySQL. All rows in a table share the same columns.
7. MongoDB's performance is better than that of MySQL and other relational DBs. This is because MongoDB sacrifices JOINS and other things and has excellent performance analysis tools. Note that you still have to index the data and the data in most applications is not enough for them to see a difference. MySQL is criticized for poor performance, especially in ORM application. However, you are unlikely to have an issue if you do proper data indexing and you are using a database wrapper.
8. One advantage of MySQL over NoSQL like MongoDB is that the community in MySQL is much better than NoSQL. This is mostly because NoSQL is relatively new while MySQL has been around for several years.
9. There are no reporting tools with MongoDB, meaning performance testing and analysis is not always possible. With MySQL, you can get several reporting tools that help you rove the validity of your applications.
10. RDBSs function on a paradigm called ACID, which is an acronym for (Atomicity, Consistency, Isolation, and Durability). This is not present in MongoDB database.
11. MongoDB has a Map Reduce feature that allows for easier scalability. This means you can get the full functionality of MongoDB database even if you are using low-cost hardware.
12. You do not have to come up with a detailed DB model with MongoDB because of is non-relational. A DB architect can quickly create a DB without a fine-grained DB model, thereby saving on development time and cost.

Q14: How many tables can a trigger associate to in MySQL? Can a trigger be associated to a view? ☆☆☆☆☆

Topics: MySQL

Answer:

A **trigger** in MySQL can be associated to a permanent table. It **can not be** associated to more than one table. Also a trigger **cannot** be associated to **temporary table** or **view**.

Q15: What happens to a trigger in MySQL if an operation which trigger is associated with fails? Does the trigger *execute*? ☆☆☆☆☆

Topics: MySQL

Answer:

A **before trigger** executes before the operation. So in case the operation fails the associated **before trigger** with execute but the **AFTER trigger** will not execute. The order of execution is **BEFORE TRIGGER,** operation and then **AFTER TRIGGER** at last. If one of them fails, it causes entire operation to fail and subsequent operation or trigger does not run. So say a **BEFORE TRIGGER** fails, the operation itself will not execute and fail. If **AFTER TRIGGER** fails, entire statement will fail.

Q16: What is difference between *horizontal* and *vertical* partitioning? Does MySQL support both horizontal and vertical partitioning? ☆☆☆☆☆

Topics: MySQL

Answer:

Partitioning is used in database to split a table data into multiple parts called partitions and store them separately. It can further be classified into horizontal partitioning and vertical partitioning.

- **Horizontal partitioning** - rows which match a partitioning function are stored in corresponding partition.
- **Vertical partitioning** - Different columns of a table can be stored in different physical partitions using vertical partitioning.

MySQL supports horizontal partitioning but **does not** support vertical partitioning.

Q17: What is the use of `SAVEPOINT` in MySQL? ☆☆☆☆☆

Topics: MySQL

Answer:

SAVEPOINT is like a bookmark while running statements in transaction mode. I helps rolling back to a specific location in the transaction. Below command can be used to create a save point while running MySQL in transaction mode.

```
SAVEPOINT savepointName;
```

And current transaction can be rolled back to desired saved save point location by running below command.

```
ROLLBACK TO savepointName;
```

Unlike **ROLLBACK** or **COMMIT** commands which end currently running transaction, **ROLLBACK TO savepointName** does not end current transaction.

Q18: Which is better for `JOIN` & `INSERT` - PostgreSQL or MySQL? ☆☆☆☆☆

Topics: MySQL PostgreSQL

Answer:

In PostgreSQL all tables are **heap tables**. In MySQL all innodb tables are **btree indexes with the tuple** in the payload. This means that primary key lookups are faster on MySQL but general queries are faster on PostgreSQL. It also means you typically need more indexes on MySQL which will slow down writes.

For example, the following query I would expect to perform better on MySQL than PostgreSQL:

```
SELECT u.username, p.*
  FROM users u
  JOIN preferences p ON u.id = p.user_id
 WHERE u.id = 123;
```

Provided that both tables share the same primary key (u.id and p.user_id), there are thousands of rows in both tables, and so forth.

On the other hand, I would expect the following query to perform better on PostgreSQL than MySQL in a db too big to fit in memory, non-cached data, proper indexes, decent-sized tables, etc:

```
SELECT c.legal_name, a.*
  FROM company c
  JOIN address a on a.company_id = c.id
 WHERE a.zip_code like '95%' and country = 'us';
```

In this case you are having to utilize other indexes, which means a lot of extra random disk I/O on MySQL.

The second issue I would expect would be write performance. I would expect PostgreSQL to generally win here due to the fact that heap tables allow inserts wherever it is convenient, and fewer indexes maintained would be helpful too.

Q19: How do you make *schema changes* to a live database without downtime? ☆☆☆☆☆

Topics: Databases PostgreSQL MySQL

Problem:

Let's say I have a PostgreSQL/MySQL database with a table including various user data like email addresses etc, all associated with specific users. If I wanted to move the email addresses to a new dedicated table, I'd have to

change the schema and then migrate the email data across to the new table. How could this be done without stopping writes to the original table?

Solution:

Follow these steps:

1. Create the new structure in parallel
2. Start writing to both structures
3. Migrate old data to the new structure
4. Only write and read new structure
5. Delete old columns

As for **step 3**, use something like this (in one transaction):

Insert what is not there yet:

```
INSERT INTO new_tbl (old_id, data)
SELECT old_id, data
FROM   old_tbl
WHERE  NOT EXISTS (SELECT * FROM new_tbl WHERE new_tbl.old_id = old_tbl.old_id);
```

Update what has changed in the meantime:

```
UPDATE new_tbl
SET    data  = old.data
USING  old_tbl
WHERE  new_tbl.old_id = old_tbl.old_id
AND    new_tbl.data IS DISTINCT FROM old_tbl.data;
```

New data will not be touched, because it is identical in both places.

Q20: Why you should never use GUIDs as part of *clustered index*?

☆☆☆☆☆

Topics: Databases MySQL PostgreSQL T-SQL

Answer:

GUIDs are not *sequential*, thus if they are part of clustered index, every time you insert new record, database would need to rearrange all its memory pages to find the right place for insertion, in case with int(bigint) auto-increment, it would be just last page.

1. MySQL - primary keys are clustered, with no option to change behavior - the recomendation is not to use GUIDs at all here
2. Postgres, MS-SQL - you can make GUID as primary key unclustered, and use another field as clustered index, for example autoincrement int.

FullStack.Cafe - Kill Your Tech Interview

Q1: Remove *duplicate rows* from a table ☆☆☆☆

Topics: MySQL

Problem:

You have a database table with duplicate rows which means certain entries are duplicate. How can you remove those entries using MySQL?

Solution:

One of the techniques which can be used for removing duplicates is by creating an intermediate table containing distinct values using **DISTINCT** command. Below syntax creates the *intermediate table*.

```
CREATE TABLE intermediate_table
SELECT DISTINCT column1, column2
FROM table_name;
```

The intermediate_table created can replace original table. Below statements drop original table and replace it with the intermediate_table.

```
DROP TABLE table_name;
```

```
ALTER TABLE intermediate_table RENAME table_name;
```