

FullStack.Cafe - Kill Your Tech Interview

Q1: What is a VIEW ? ☆

Topics: SQL

Answer:

A **view** is simply a virtual table that is made up of elements of multiple physical or “real” tables. Views are most commonly used to join multiple tables together, or control access to any tables existing in background server processes.

Q2: What is PRIMARY KEY ? ☆

Topics: SQL

Answer:

- A **PRIMARY KEY** constraint is a *unique identifier* for a row within a database table.
- Every table *should have* a primary key constraint to uniquely identify each row and only one primary key constraint can be created for each table.
- The primary key constraints are used to enforce *entity integrity*.

Q3: 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;
```

Q4: What is FOREIGN KEY ? ☆☆☆

Topics: SQL

Answer:

- **FOREIGN KEY** constraint *prevents* any actions that would destroy links between tables with the corresponding data values.
- A foreign key in one table *points* to a primary key in another table.
- Foreign keys prevent actions that would leave rows with foreign key values when there are no primary keys with that value.
- The foreign key constraints are used to **enforce referential integrity**.

Q5: 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.

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: 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.

Q8: 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);
```

Q9: 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';
```

Q10: Discuss INNER JOIN ON vs WHERE clause (with multiple FROM tables) ☆☆☆

Topics: SQL

Problem:

You can do:

```
SELECT
    table1.this, table2.that, table2.somethingelse
FROM
    table1, table2
WHERE
    table1.foreignkey = table2.primarykey
    AND (some other conditions)
```

Or else:

```
SELECT
    table1.this, table2.that, table2.somethingelse
FROM
    table1 INNER JOIN table2
    ON table1.foreignkey = table2.primarykey
WHERE
    (some other conditions)
```

What syntax would you choose and why?

Solution:

- **INNER JOIN** is ANSI syntax that you should use. INNER JOIN helps human readability, and that's a top priority. It can also be easily replaced with an OUTER JOIN whenever a need arises.
- *Implicit* joins (with multiple **FROM** tables) become much much more confusing, hard to read, and hard to maintain once you need to start adding more tables to your query. The old syntax, with just listing the tables, and using the **WHERE** clause to specify the join criteria, is being deprecated in most modern databases.

Q11: 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.

Q12: 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');
```

Q13: 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.

Q14: What is the difference between **INNER JOIN**, **OUTER JOIN**, **FULL OUTER JOIN**? ☆☆☆

Topics: SQL

Answer:


An **inner join** retrieve the matched rows only.

Whereas an **outer join** retrieve the matched rows from one table and all rows in other tablethe result depends on which one you are using:

- **Left:** Matched rows in the right table and all rows in the left table
- **Right:** Matched rows in the left table and all rows in the right table or
- **Full:** All rows in all tables. It doesn't matter if there is a match or not

Inner Join

Retrieve the matched rows only, that is, `A intersect B`.

 Enter image description here

```
SELECT *
FROM dbo.Students S
INNER JOIN dbo.Advisors A
    ON S.Advisor_ID = A.Advisor_ID
```

Left Outer Join

Select all records from the first table, and any records in the second table that match the joined keys.

 Enter image description here

```
SELECT *
FROM dbo.Students S
LEFT JOIN dbo.Advisors A
    ON S.Advisor_ID = A.Advisor_ID
```

Full Outer Join

Select all records from the second table, and any records in the first table that match the joined keys.

 Enter image description here

```
SELECT *
FROM dbo.Students S
FULL JOIN dbo.Advisors A
    ON S.Advisor_ID = A.Advisor_ID
```

Q15: 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
```

Q16: What is the difference between INNER JOIN and OUTER JOIN ?



Topics: SQL

Problem:

Also, how do `LEFT JOIN` , `RIGHT JOIN` and `FULL JOIN` fit in?

Solution:

Assuming you're joining on columns with no duplicates, which is a very common case:

- An **INNER JOIN** of A and B gives the result of A intersect B, i.e. the inner part of a Venn diagram intersection.
- An **OUTER JOIN** of A and B gives the results of A union B, i.e. the outer parts of a Venn diagram union.
- A **LEFT OUTER JOIN** will give all rows in A, plus any common rows in B.
- A **RIGHT OUTER JOIN** will give all rows in B, plus any common rows in A.
- A **FULL OUTER JOIN** will give you the **union** of A and B, i.e. all the rows in A and all the rows in B. If something in A doesn't have a corresponding datum in B, then the B portion is null, and vice versa.

Q17: How to select first 5 records from a table? ☆☆☆

Topics: SQL

Answer:

```
-- SQL Server
SELECT TOP 5 * FROM EMP;
```

```
-- Oracle
SELECT * FROM EMP WHERE ROWNUM <= 5;
```

```
-- Generic
SELECT name FROM EMPLOYEE o
       WHERE (SELECT count(*) FROM EMPLOYEE i WHERE i.name < o.name) < 5
```

Q18: What is the difference between WHERE clause and HAVING clause? ☆☆☆

Topics: SQL

Answer:

WHERE clause introduces a condition on *individual rows*; HAVING clause introduces a condition on *aggregations*, i.e. results of selection where a single result, such as count, average, min, max, or sum, has been produced from *multiple* rows. Your query calls for a second kind of condition (i.e. a condition on an aggregation) hence HAVING works correctly.

As a rule of thumb, use WHERE before GROUP BY and HAVING after GROUP BY. It is a rather primitive rule, but it is useful in more than 90% of the cases.

While you're at it, you may want to re-write your query using ANSI version of the join:

```
SELECT L.LectID, Fname, Lname
FROM Lecturers L
JOIN Lecturers_Specialization S ON L.LectID=S.LectID
GROUP BY L.LectID, Fname, Lname
HAVING COUNT(S.Expertise)>=ALL
        (SELECT COUNT(Expertise) FROM Lecturers_Specialization GROUP BY LectID)
```

This would eliminate WHERE that was used as a *theta join condition*.

Q19: What is the difference between UNION and UNION ALL? ☆☆☆

Topics: SQL

Answer:

UNION removes duplicate records (where all columns in the results are the same), UNION ALL does not.

There is a performance hit when using UNION instead of UNION ALL, since the database server must do additional work to remove the duplicate rows, but usually you do not want the duplicates (especially when developing reports).

UNION Example:

```
SELECT 'foo' AS bar UNION SELECT 'foo' AS bar
```

Result:


```
+-----+
| bar |
+-----+
| foo |
+-----+
1 row in set (0.00 sec)
```

UNION ALL example:

```
SELECT 'foo' AS bar UNION ALL SELECT 'foo' AS bar
```

Result:

```
+-----+
| bar |
+-----+
| foo |
| foo |
+-----+
2 rows in set (0.00 sec)
```

Q20: What is the difference between JOIN and UNION ? ☆☆☆**Topics:** SQL**Answer:**

UNION puts lines from queries after each other, while JOIN makes a *cartesian* product and subsets it -- completely different operations. Trivial example of UNION :

```
mysql> SELECT 23 AS bah
-> UNION
-> SELECT 45 AS bah;
+-----+
| bah |
+-----+
| 23 |
| 45 |
+-----+
2 rows in set (0.00 sec)
```

similary trivial example of JOIN :

```
mysql> SELECT * FROM
-> (SELECT 23 AS bah) AS foo
-> JOIN
-> (SELECT 45 AS bah) AS bar
-> ON (33=33);
+-----+-----+
| foo | bar |
+-----+-----+
| 23 | 45 |
+-----+-----+
1 row in set (0.01 sec)
```