**Q. What is a database?**

A database is a systematic or organized collection of related information that is stored in such a way that it can be easily accessed, retrieved, managed, and updated.

**Syntax:**

CREATE DATABASE <databaseName>

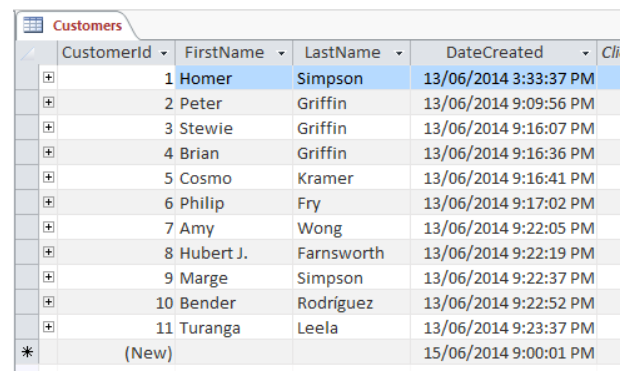
**Example:**

CREATE DATABASE Product

**Q. What is a database table?**

A database table is a structure that organises data into rows and columns – forming a grid.

Tables are similar to a worksheets in spreadsheet applications. The rows run horizontally and represent each record. The columns run vertically and represent a specific field. The rows and columns intersect, forming a grid. The intersection of the rows and columns defines each cell in the table.

[](https://github.com/learning-zone/sql-basics/blob/master/assets/table.png)

**Syntax:**

CREATE TABLE <table\_name> (ID INT, NAME VARCHAR(30) )

DROP TABLE <table\_name>

SELECT \* FROM <table\_name>

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. What is a database relationship?**

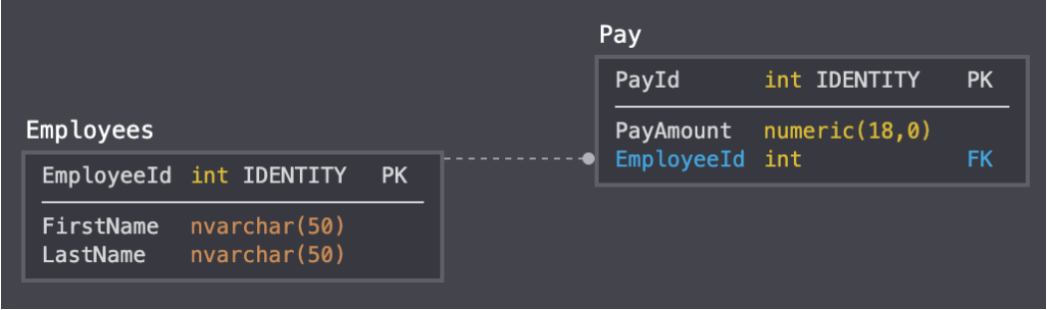
Database relationships are associations between tables that are created using join statements to retrieve data. It helps improve table structures and reduce redundant data.

Understanding relationship in databases is important as it allows you to fetch data from multiple tables simultaneously and helps ensure that data in databases are consistent and updated.

**Types of Database Relationships:**

**1. One-to-One:**

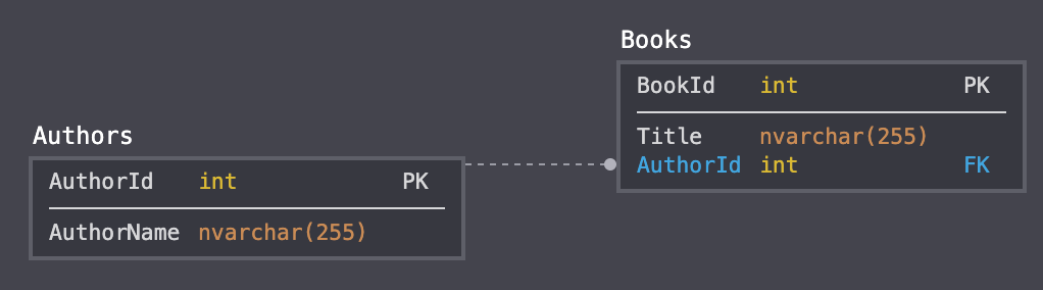
A one-to-one relationship is a relationship between two tables where each table can have only one matching row in the other table.

[](https://github.com/learning-zone/sql-basics/blob/master/assets/one-to-one.png)

Using the above screenshot as an example, the business case is that each employee's pay details must be stored in a separate table to the employee's contact details. In such a case, there can only be one row in the Pay table that matches a given employee in the Employees table. This is a good candidate for a one-to-one relationship.

**2. One-to-Many:**

The one-to-many relationship is similar to the one-to-one relationship, except that it allows multiple matching rows in one of the tables.

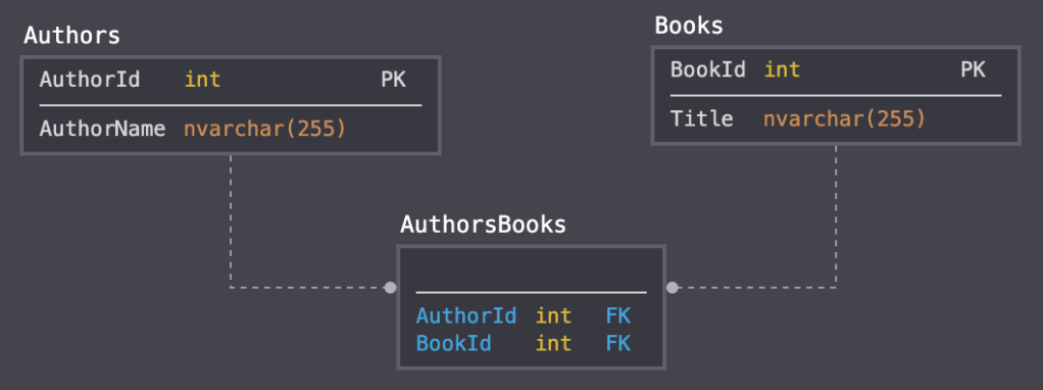
[](https://github.com/learning-zone/sql-basics/blob/master/assets/one-to-many.png)

In the above example, each author can have many books, but each book can only have one author.

Therefore, the Books table is allowed to contain multiple rows with the same AuthorId value. If an author has released five books, then there would be one row in Authors for that author, and five rows in Books, each with that author's AuthorId.

**3. Many-to-Many:**

In a many-to-many relationship, each side of the relationship can contain multiple rows.

[](https://github.com/learning-zone/sql-basics/blob/master/assets/many-to-many.png)

In this example, each book is allowed to have multiple authors. Therefore, I created a lookup table (also known as a "junction table") that stores both the AuthorId and the BookId.

These two columns could be configured to be the primary key of the table (in which case they would be a "composite primary key" or simply "composite key"), or you could create a separate column to be the primary key.

Note that the Books table doesn't have AuthorId in this case. That column has been moved to the AuthorBooks table so that we can have many AuthorIds for the same BookId.

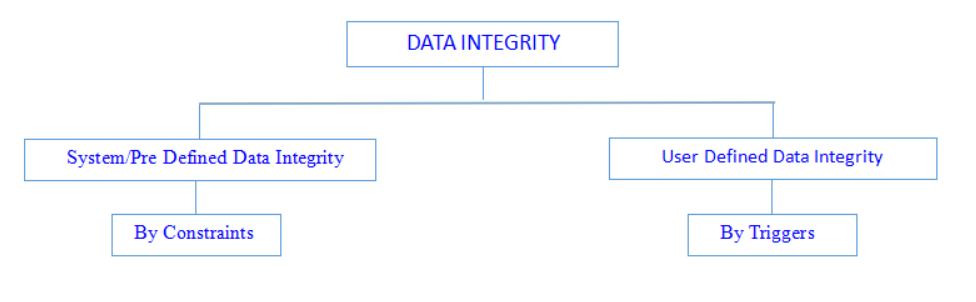
[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. What is data Integrity?**

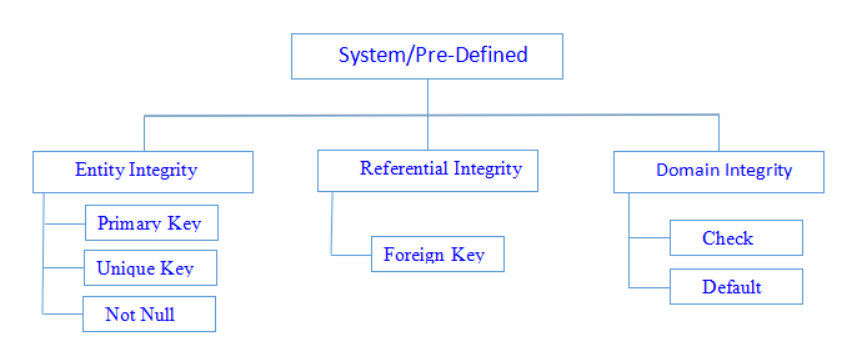
Data Integrity defines the accuracy and consistency of data stored in a database. It can also define integrity constraints to enforce business rules on the data when it is entered into the application or database.

**Classification of Data Integrity:**

* a) System/Pre Defined Integrity
* b) User-Defined Integrity

[](https://github.com/learning-zone/sql-basics/blob/master/assets/data-integrity.png)

**a) System/Pre Defined Integrity:**

[](https://github.com/learning-zone/sql-basics/blob/master/assets/system-integrity.png)

**1. Entity Integrity:**

Entity integrity ensures each row in a table is a uniquely identifiable entity. We can apply Entity integrity to the Table by specifying a primary key, unique key, and not null.

**2. Referential Integrity:**

Referential integrity ensures the relationship between the Tables.

We can apply this using a Foreign Key constraint.

**3. Domain Integrity:**

Domain integrity ensures the data values in a database follow defined rules for values, range, and format. A database can enforce these rules using Check and Default constraints.

**b) User-Defined Integrity:**

It comprises the rules defined by the operator to fulfill their specific requirements. Entity, referential, and domain integrity are not enough to refine and secure data. Time in time again, particular business rules must be considered and integrated into data integrity processes to meet enterprise standards.

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. What are the two principles of relational database model?**

The two principal rules for the relational model are as follows:

* Entity integrity: this is used to maintain the integrity at entity level
* Referential integrity: it is used to maintain integrity on all the values which have been referenced.

The differences between them are as follows:

* Entity integrity tells that in a database every entity should have a unique key; on the other hand referential integrity tells that in the database every table values for all foreign keys will remain valid.
* Referential integrity is based on entity integrity but it is not the other way around.
* For example: if a table is present and there is a set of column out of which one column has parent key set then to ensure that the table doesn'\t contain any duplicate values, a unique index is defined on the column that contains the parent key.

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. What are the relational operations that can be performed on the database?**

There are many relational operators that are used to perform actions on relational database. These operators are as follows:

1. Union operator that combines the rows of two relations and doesn'\t include any duplicate. It also removes the duplicates from the result.
2. Intersection operator provides a set of rows that two relations have in common.
3. Difference operator provide the output by taking two relations and producing the difference of rows from first that don'\t exist in second.
4. Cartesian product is done on two relations. It acts as a cross join operator.

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. What do you understand by database Normalization?**

* Normalization is very essential part of relational model.
* Normal forms are the common form of normalization.
* It helps in reducing redundancy to increase the information overall.
* It has some disadvantages as it increases complexity and have some overhead of processing.
* It consists of set of procedures that eliminates the domains that are non-atomic and redundancy of data that prevents data manipulation and loss of data integrity.

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. What are the different types of normalization that exists in the database?**

There are 9 normalizations that are used inside the database. These are as follows:

1. First normal form: in this table represents a relation that has no repeating groups.
2. Second normal form: non- prime attributes are not functional dependent on subset of any candidate key.
3. Third normal form: in a table every non- prime attribute is non-transitively dependent on every candidate key
4. Elementary key normal form: superkey dependency or elementary key dependency effects the functional dependency in a table.
5. Boyce codd normal form: “every non-trivial functional dependency in the table is dependent on superkey”.
6. Fourth normal form: “Every non-trivial multivalued dependency in the table is a dependent on a superkey”.
7. Fifth normal form (5NF): “Every non-trivial join dependency in the table is implied by the superkeys of the table”.
8. Domain/key normal form (DKNF): “Every constraint on the table is a logical consequence of the table's domain constraints and key constraints”.
9. Sixth normal form (6NF): “Table features no non-trivial join dependencies at all”.

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. How de-normalization is different from normalization?**

* Analytical processing databases are not very normalized. The operations which are used are read most databases.
* It is used to extract the data that are ancient and accumulated over long period of time. For this purpose de-normalization occurs that provide smart business applications.
* Dimensional tables in star schema are good example of de-normalized data.
* The de-normalized form must be controlled while extracting, transforming, loading and processing.
* There should be constraint that user should not be allowed to view the state till it is consistent.
* It is used to increase the performance on many systems without RDBMS platform.

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. What is the type of de-normalization?**

Non-first normal form (NFA)

– It describes the definition of the database design which is different from the first normal form.

* It keeps the values in structured and specialized types with their own domain specific languages.
* The query language used in this is extended to incorporate more support for relational domain values by adding more operators.

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. How many levels of data abstraction are available?**

There are three levels of data abstraction available in database model and these are as follows:

1. Physical level: It is the lowest level that describes how data is stored inside the database.
2. Logical level: It is the next higher level in the hierarchy that provides the abstraction. It describes what data are stored and the relationship between them.
3. View level: It is the highest level in hierarchy that describes part of the entire database. It allows user to view the database and do the query.

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. What do you understand by Data Independence?**

Data independence tells about the independence of the data inside the application. It usually deals with the storage structure and represents the ability to modify the schema definition. It doesn'\t affect the schema definition which is being written on the higher level.

There are two types of data independence:

1. Physical data independence: It allows the modification to be done in physical level and doesn'\t affect the logical level.
2. Logical data independence: It allow the modification to be done at logical level and affects the view level.

NOTE: Logical Data Independence is more difficult to achieve.

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. How view is related to data independence?**

* View is a virtual table that doesn'\t really exist, but it remains present so that user can view their data.
* It is derived from the base table. The view is stored in the data dictionary and represents the file directly.
* The base table updation or reconstruction is not being reflected in views.
* It is related to the logical data independence as it is at the logical level and not at the physical level.

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. Why E-R models are used?**

E-R model stands for entity-relationship model and it is used to represent a model with their relationships. This is an object oriented approach and it is based on real world that consists of objects which are called entities and relationship between them. Entities are further used inside the database in the form of attributes.

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. What do you understand by cardinality and why it is used?**

* Cardinality is important and used to arrange the data inside the database.
* It is related to the design part and need to be properly used in database.
* It is used in E-R diagrams and used to show the relationship between entities/tables.
* It has many forms like the basic is one to one, which associate one entity with another.
* Second is one to many: which relates one entity with many entities in a table.
* Third is many to many M: N that allows many entities to be related to many more.
* Last is many to one that allows the many entities to be associated with one entity.

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. What is DDL, DML and DCL?**

SQL commands can be divided in three large subgroups.

1. DDL: The SQL commands which deals with database schemas and information of how the data will be generated in database are classified as Data Definition Language. -For example: CREATE TABLE or ALTER TABLE belongs to DDL.
2. DML: The SQL commands which deals with data manipulation are classified as Data Manipulation Language. For example: SELECT, INSERT, etc.
3. DCL: The SQL commands which deal with rights and permission over the database are classified as DCL. For example: GRANT, REVOKE

**Q. How to prevent from database SQL Injection?**

SQL Injection is a code-based vulnerability that allows an attacker to read and access sensitive data from the database. Attackers can bypass security measures of applications and use SQL queries to modify, add, update, or delete records in a database.

**Simple SQL Injection Example:**

SELECT id FROM users WHERE username='username' AND password='password' OR 1=1'

Because of the **OR 1=1** statement, the **WHERE** clause returns the first **id** from the **users** table no matter what the **username** and **password** are. The first user id in a database is very often the administrator. In this way, the attacker not only bypasses authentication but also gains administrator privileges.

**Prevent SQL Injections:**

**1. Continuous Scanning and Penetration Testing:**

The automated web application scanner has been the best choice to point out vulnerabilities within the web applications for quite some time now. Now, with SQL injections getting smarter in exploiting logical flaws, website security professionals should explore manual testing with the help of a security vendor.

They can authenticate user inputs against a set of rules for syntax, type, and length. It helps to audit application vulnerabilities discreetly so that you can patch the code before hackers exploit it to their advantage.

**2. Restrict Privileges:**

It is more of a database management function, but enforcing specific privileges to specific accounts helps prevent blind SQL injection attacks. Begin with no privileges account and move on to "read-only", "edit", "delete" and similar privilege levels.

Minimizing privileges to the application will ensure that the attacker, who gets into the database through the application, cannot make unauthorized use of specific data.

**3. Use Query Parameters:**

Dynamic queries create a lot of troubles for security professionals. They have to deal with variable vulnerabilities in each application, which only gets graver with updates and changes. It is recommended that you prepare parameterized queries.

These queries are simple, easy to write, and only pass when each parameter in SQL code is clearly defined. This way, your info is supplied with weapons to differentiate between code and information inputs.

**4. Instant Protection:**

A majority of organizations fail the problems like outdated code, scarcity of resources to test and make changes, no knowledge of application security, and frequent updates in the application. For these, web application protection is the best solution.

A managed web application firewall can be deployed for immediate mitigation of such attacks. It contains custom policies to block any suspicious input and deny information breach instantly. This way, you do not have to manually look for loopholes and mend problems afterward.

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. What are the non standard string types available in SQL?**

Following are Non-Standard string types:

| **Name** | **Max Length** |
| --- | --- |
| TINYTEXT | 255 bytes |
| TEXT | 65,535 bytes |
| MEDIUMTEXT | 16 MB |
| LONGTEXT | 4GB |

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**# 2. SQL Data Types**

**Q. What is difference between CHAR and VARCHAR in MySQL?**

Both of them are used for string type data. char has fixed length and if the inserted data is less than the defined length, required no. of blank spaces are added as padding. varchar has variable length and no padding is used to fill up the left out space. So technically, varchar will save space.

**Q. What are the string datatypes in SQL?**

A list of data types used in MySQL database. This is based on MySQL 8.0.

| **Data Types** | **Description** |
| --- | --- |
| CHAR(Size) | It is used to specify a fixed length string that can contain numbers, letters, and special characters. Its size can be 0 to 255 characters. Default is 1. |
| VARCHAR(Size) | It is used to specify a variable length string that can contain numbers, letters, and special characters. Its size can be from 0 to 65535 characters. |
| BINARY(Size) | It is equal to CHAR() but stores binary byte strings. Its size parameter specifies the column length in the bytes. Default is 1. |
| VARBINARY(Size) | It is equal to VARCHAR() but stores binary byte strings. Its size parameter specifies the maximum column length in bytes. |
| TEXT(Size) | It holds a string that can contain a maximum length of 255 characters. |
| TINYTEXT | It holds a string with a maximum length of 255 characters. |
| MEDIUMTEXT | It holds a string with a maximum length of 16,777,215. |
| LONGTEXT | It holds a string with a maximum length of 4,294,967,295 characters. |
| ENUM(val1, val2, val3,...) | It is used when a string object having only one value, chosen from a list of possible values. It contains 65535 values in an ENUM list. If you insert a value that is not in the list, a blank value will be inserted. |
| SET( val1,val2,val3,....) | It is used to specify a string that can have 0 or more values, chosen from a list of possible values. You can list up to 64 values at one time in a SET list. |
| BLOB(size) | It is used for BLOBs (Binary Large Objects). It can hold up to 65,535 bytes. |

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. What are the differences between the BLOB and TEXT datatypes in MySQL?**

BLOB stands for Binary Large Objects and as its name suggests, it can be used for storing binary data while TEXT is used for storing large number of strings. BLOB can be used to store **binary data** that means we can store pictures, videos, sounds and programs also.

BLOB values behave like byte string and BLOB does not have a character set. Therefore, comparison and sorting is fully dependent upon numeric values of bytes.

TEXT values behave like non-binary string or character string. TEXT has a character set and the comparison/ sorting fully depends upon the collection of character set.

**Creating a table with TEXT data type:**

mysql> create table TextTableDemo ( Address TEXT );

mysql> DESC TextTableDemo;

**Creating a table with BLOB type:**

mysql> create table BlobTableDemo ( Images BLOB );

mysql> desc BlobTableDemo;

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**# 3. SQL Database**

**Q. How to create a database using SQL?**

**# 4. SQL Table**

**Q. How to create a table in SQL?**

**Q. What are tables and Fields?**

**Q. How to delete a table in SQL Server?**

**Q. What is the difference between DELETE TABLE and TRUNCATE TABLE commands?**

**Q. What is the difference between TRUNCATE and DROP statements?**

**Q. How to alter a table schema in SQL Server?**

**Q. What are Heap tables in SQL?**

*ToDo*

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**# 5. SQL Select**

**Q. What are query types in a database?**

**Q. What is the difference between UNION and UNION ALL?**

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. What is difference between Correlated subquery and nested subquery?**

**1. Correlated subqueries:**

Correlated subqueries are used for row-by-row processing. Each subquery is executed once for every row of the outer query.

A correlated subquery is evaluated once for each row processed by the parent statement. The parent statement can be a SELECT, UPDATE, or DELETE statement.

**Example:**

--- Correlated Subquery

SELECT e.EmpFirstName, e.Salary, e.DeptId

FROM Employee e

WHERE e.Salary = (SELECT max(Salary) FROM Employee ee WHERE ee.DeptId = e.DeptId)

**2. Nested subqueries:**

A subquery can be nested inside other subqueries. SQL has an ability to nest queries within one another. A subquery is a SELECT statement that is nested within another SELECT statement and which return intermediate results. SQL executes innermost subquery first, then next level.

**Example:**

--- Nested Subquery

SELECT EmpFirstName, Salary, DeptId

FROM Employee

WHERE (DeptId, Salary) IN (SELECT DeptId, max(Salary) FROM Employee group by DeptId)

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. Differentiate UNION, MINUS, UNION ALL and INTERSECT?**

* INTERSECT - It will give all the distinct rows from both select queries.
* MINUS - It will give distinct rows returned by the first query but not by the second query.
* UNION - It will give all distinct rows selected by either first query or second query.
* UNION ALL - It will give all rows returned by either query with all duplicate records.

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. Explain SQL Operators?**

| **Sl.No** | **Query** | **Description** |
| --- | --- | --- |
| 01. | SELECT c1 FROM t1 UNION [ALL] SELECT c1 FROM t2 | Select column c1 from a table named t1 and column c1 from a table named t2 and combine the rows from these two queries |
| 02. | SELECT c1 FROM t1 INTERSECT SELECT c1 FROM t2 | Select column c1 from a table named t1 and column c1 from a table named t2 and return the intersection of two queries |
| 03. | SELECT c1 FROM t1 MINUS SELECT c1 FROM t2 | Select column c1 from a table named t1 and column c1 from a table named t2 and subtract the 2nd result set from the 1st |
| 04. | SELECT c1 FROM t WHERE c1 [NOT] LIKE pattern | Select column c1 from a table named t and query the rows using pattern matching % |
| 05. | SELECT c1 FROM t WHERE c1 [NOT] in test\_list | Select column c1 from a table name t and return the rows that are (or are not) in test\_list |
| 06. | SELECT c1 FROM t WHERE c1 BETWEEN min AND max | Select column c1 from a table named t and return the rows where c1 is between min and max |
| 07. | SELECT c1 FROM t WHERE c1 IS [NOT] NULL | Select column c1 from a table named t and check if the values are NULL or not |

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. Explain correlated query work?**

**Q. What is the SQL CASE statement used for?**

*ToDo*

**# 6. SQL Clause**

**Q. What is the difference between a HAVING CLAUSE and a WHERE CLAUSE?**

*ToDo*

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**# 7. SQL Order By**

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**# 8. SQL Insert**

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**# 9. SQL Update**

**Q. What are COMMIT and ROLLBACK in SQL?**

**COMMIT** statement is used to end the current transaction and once the COMMIT statement is exceucted the transaction will be permanent and undone.

**Example:**:

BEGIN

UPDATE EmpDetails SET EmpName = "Arpit" where Dept = "Developer"

COMMIT;

END;

**ROLLBACK** statement is used to end the current transaction and undone the changes which was made by that transaction.

**Syntax:** ROLLBACK [TO] Savepoint\_name;

**Example:**:

BEGIN

Statement1;

SAVEPOINT mysavepoint;

BEGIN

Statement2;

EXCEPTION

WHEN OTHERS THEN

ROLLBACK TO mysavepoint;

Statement5;

END;

END;

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. Explain data modification commands in SQL?**

| **Sl.No** | **Query** | **Description** |
| --- | --- | --- |
| 01. | INSERT INTO t(column\_list) VALUES(value\_list) | Insert one row into a table named t |
| 02. | INSERT INTO t(column\_list) VALUES (value\_list), (value\_list), … | Insert multiple rows into a table named t |
| 03. | INSERT INTO t1(column\_list) SELECT column\_list FROM t2 | Insert rows from t2 into a table named t1 |
| 04. | UPDATE tSET c1 = new\_value | Update a new value in table t in the column c1 for all rows |
| 05. | UPDATE tSET c1 = new\_value, c2 = new\_value WHERE condition | Update values in column c1 and c2 in table t that match the condition |
| 06. | DELETE FROM t | Delete all the rows from a table named t |
| 07. | DELETE FROM tWHERE condition | Delete all rows from that a table named t that match a certain condition |

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**# 10. SQL Delete**

**Q. What is difference between Truncate and Delete in SQL?**

* TRUNCATE is a DDL (data definition language) command whereas DELETE is a DML (data manipulation language) command.
* We can'\t execute a trigger with TRUNCATE whereas with DELETE command, a trigger can be executed.
* We can use any condition in WHERE clause using DELETE but it is not possible with TRUNCATE.
* If table is referenced by any foreign key constraints then TRUNCATE cannot work.
* TRUNCATE is faster than DELETE, because when you use DELETE to delete the data, at that time it store the whole data in rollback space from where you can get the data back after deletion, whereas TRUNCATE will not store data in rollback space and will directly delete it. You can'\t get the deleted data back when you use TRUNCATE.

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**# 11. SQL Keys**

**Q. What is the difference between primary and foreign key?**

* Primary key uniquely identify a relationship in a database, whereas foreign key is the key that is in other relation and it has been referenced from the primary key from other table.
* Primary key remains one only for the table, whereas there can be more than one foreign key.
* Primary key is unique and won'\t be shared between many tables, but foreign key will be shared between more than one table and will be used to tell the relationship between them.

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. What is a unique key?**

**Q. What is a foreign key of a database?**

**Q. What is a constraint in SQL?**

**Q. How do I define constraints in SQL?**

**Q. What is a candidate key?**

**Q. What is the default index created on primary key in sql server?**

*ToDo*

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**# 12. SQL Join**

**Q. Explain JOIN Query in mySQL?**

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.  
Take users table and orders table for example.

**Users Table:**

| **user\_id** | **name** | **mobile** |
| --- | --- | --- |
| 1 | John | 123 |
| 2 | Joe | 124 |

**Orders Table:**

| **order\_id** | **user\_id** | **total** | **created\_at** |
| --- | --- | --- | --- |
| 1 | 1 | 500 | 2022-12-19 18:32:00 |
| 2 | 1 | 800 | 2021-12-03 08:32:00 |
| 3 | 2 | 50 | 2020-12-13 12:49:00 |
| 4 | 1 | 80 | 2021-12-15 21:19:00 |

So to get the list of orders with names and mobile nos. for each order, we can join orders and users on the basis of user\_id.

SELECT

o.\*,

u.name,

u.mobile

FROM

ordes o

JOIN users u ON o.user\_id = u.user\_id;

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. Explain the different types of joins?**

Using Join in a query, we can retrieve referenced columns or rows from multiple tables.

Following are different types of Joins:

1. JOIN: Return details from tables if there is at least one matching row in both tables.
2. LEFT JOIN: It will return all rows from the left table, even if there are no matching row in the right table.
3. RIGHT JOIN: It will return all rows from the right table, even if there is no matching row in the left table.
4. FULL JOIN: It will return rows when there is a match in either of tables.

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. What are Self Join and Cross Join?**

* When we want to join a table to itself then SELF JOIN is used.
* We can give one or more aliases to eliminate the confusion.
* A self join can be used as any type, if both the tables are same.
* The simple example where we can use SELF JOIN is if in a company have a hierarchal reporting structure and an employee reports to another.
* A cross join give the number of rows in the first table multiplied by the number of rows in second table.
* The simple example where we can use CROSS JOIJ is if in an organization wants to combine every Employee with family table to see each Employee with each family member.

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. How to query data from multiple tables?**

| **Sl.No** | **Query** | **Description** |
| --- | --- | --- |
| 01. | SELECT c1, c2 FROM t1 INNER JOIN t2 on condition | Select columns c1 and c2 from a table named t1 and perform an inner join between t1 and t2 |
| 02. | SELECT c1, c2 FROM t1 LEFT JOIN t2 on condition | Select columns c1 and c2 from a table named t1 and perform a left join between t1 and t2 |
| 03. | SELECT c1, c2 FROM t1 RIGHT JOIN t2 on condition | Select columns c1 and c2 from a table named t1 and perform a right join between t1 and t2 |
| 04. | SELECT c1, c2 FROM t1 FULL OUTER JOIN t2 on condition | Select columns c1 and c2 from a table named t1 and perform a full outer join between t1 and t2 |
| 05. | SELECT c1, c2 FROM t1 CROSS JOIN t2 | Select columns c1 and c2 from a table named t1 and produce a Cartesian product of rows in tables |
| 06. | SELECT c1, c2 FROM t1, t2 | Select columns c1 and c2 from a table named t1 and produce a Cartesian product of rows in tables |
| 07. | SELECT c1, c2 FROM t1 A INNER JOIN t2 B on condition | Select columns c1 and c2 from a table named t1 and joint it to itself using an INNER JOIN clause |

**Q. What is full join in SQL?**

**Q. What is an outer join in SQL?**

**Q. What is an inner join in SQL?**

**Q. What is left join in SQL Server?**

**Q. What is a right join in SQL Server?**

**Q. What is the default join in SQL?**

*ToDo*

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**# 13. SQL RegEx**

**Q. How to use REGEXP in SQL Query?**

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**# 14. SQL Indexes**

**Q. What are indexes in a Database?**

Indexing is a way to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed. It is a data structure technique which is used to quickly locate and access the data in a database.

Indexes are created using a few database columns

* The first column is the **Search key** that contains a copy of the primary key or candidate key of the table. These values are stored in sorted order so that the corresponding data can be accessed quickly.
* The second column is the **Data Reference** or **Pointer** which contains a set of pointers holding the address of the disk block where that particular key value can be found.

| **Sl.No** | **Query** | **Description** |
| --- | --- | --- |
| 01. | CREATE INDEX index\_name ON t(c1, c2) | Create an index on columns c1 and c2 of the table t |
| 02. | CREATE UNIQUE INDEX index\_name ON t(c3, c4) | Create a unique index on columns c3 and c4 of the table t |
| 03. | DROP INDEX index\_name | Drop an index |

**Example:**

-- Create Index

CREATE INDEX <index\_name> ON <table\_name> (column1, column2, ...)

-- Show Index

SHOW INDEX FROM <table\_name>;

-- Alter Index

ALTER TABLE <table\_name> ADD INDEX(`column\_name`);

-- Drop Index

DROP INDEX index\_name ON <table\_name>;

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. What is an index represent in relational database model?**

Index is a way to provide quick access to the data and structure. It has indexes maintain and can be created to combine attributes on a relation. Index allows the queries to filter out the searches faster and matching data can be found earlier with simplicity.

For example: It is same as the book where by using the index you can directly jump to a defined section. In relational database there is a provision to give multiple indexing techniques to optimize the data distribution.

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. What is the difference between Cluster and Non-Cluster Index?**

**Q. How to create index in SQL Server?**

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. What are the types of indexes in sql?**

**1. Clustered Index:**

Clustered index is the type of indexing that establishes a physical sorting order of rows. Clustered index is like Dictionary; in the dictionary, sorting order is alphabetical and there is no separate index page.

**2. Non-clustered:**

Non-Clustered index is an index structure separate from the data stored in a table that reorders one or more selected columns. The non-clustered index is created to improve the performance of frequently used queries not covered by a clustered index. It's like a textbook; the index page is created separately at the beginning of that book.

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**# 15. SQL Wildcards**

**Q. What are the ways to use wildcards in sql?**

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**# 16. SQL Date Format**

**Q. What is difference between timestamp and datetime in SQL?**

*ToDo*

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**# 17. SQL Transactions**

**Q. What is transactions in SQL?**

A SQL transaction is a grouping of one or more SQL statements that interact with a database. A transaction in its entirety can commit to a database as a single logical unit or rollback (become undone) as a single logical unit.

In SQL, transactions are essential for maintaining database integrity. They are used to preserve integrity when multiple related operations are executed concurrently, or when multiple users interact with a database concurrently.

**Properties of Transactions:**

Transactions have the following four standard properties, usually referred to by the acronym ACID.

* **Atomicity** − ensures that all operations within the work unit are completed successfully. Otherwise, the transaction is aborted at the point of failure and all the previous operations are rolled back to their former state.
* **Consistency** − ensures that the database properly changes states upon a successfully committed transaction.
* **Isolation** − enables transactions to operate independently of and transparent to each other.
* **Durability** − ensures that the result or effect of a committed transaction persists in case of a system failure.

**Transaction Control:**

The following commands are used to control transactions.

* **COMMIT** − to save the changes.
* **ROLLBACK** − to roll back the changes.
* **SAVEPOINT** − creates points within the groups of transactions in which to ROLLBACK.
* **SET TRANSACTION** − Places a name on a transaction.

**Example:**

-- Exmaple - 01

CREATE TABLE widgetInventory (

id SERIAL,

description VARCHAR(255),

onhand INTEGER NOT NULL

);

CREATE TABLE widgetSales (

id SERIAL,

inv\_id INTEGER,

quan INTEGER,

price INTEGER

);

INSERT INTO widgetInventory ( description, onhand ) VALUES ( 'rock', 25 );

INSERT INTO widgetInventory ( description, onhand ) VALUES ( 'paper', 25 );

INSERT INTO widgetInventory ( description, onhand ) VALUES ( 'scissors', 25 );

START TRANSACTION;

INSERT INTO widgetSales ( inv\_id, quan, price ) VALUES ( 1, 5, 500 );

UPDATE widgetInventory SET onhand = ( onhand - 5 ) WHERE id = 1;

COMMIT;

SELECT \* FROM widgetInventory;

SELECT \* FROM widgetSales;

START TRANSACTION;

INSERT INTO widgetInventory ( description, onhand ) VALUES ( 'toy', 25 );

ROLLBACK;

SELECT \* FROM widgetInventory;

SELECT \* FROM widgetSales;

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. What is the purpose of acid properties?**

* ACID stands for Atomicity, Consistency, Isolation and durability and it plays an important role in the database.
* These properties allow the database to be more convenient to access and use. This allows data to be shared more safely in between the tables.
* If these properties are not being implemented then the data will become inconsistent and inaccurate.
* It helps in maintaining the accuracy of the data in the database.

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. What is a NOLOCK?**

* NOLOCK is used to improve concurrency on a busy system.
* On data read, no lock can be taken on SELECT statement.
* When some other process is updating the data on the same time you are reading it is known as dirty read.
* Read (Shared) locks are taken by SELECT Statements.
* Simultaneous access of multiple SELECT statements is allowed in Shared lock but modification process is not allowed.
* The result to your system is blocking.
* Update will start on completion of all the reads.

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. What is a WITH(NOLOCK)?**

* WITH(NOLOCK) is used to unlock the data which is locked by the transaction that is not yet committed. This command is used before SELECT statement.
* When the transaction is committed or rolled back then there is no need to use NOLOCK function because the data is already released by the committed transaction.
* Syntax: WITH(NOLOCK)

**Example:**:

SELECT \* FROM EmpDetails WITH(NOLOCK)

WITH(NOLCOK) is similar as READ UNCOMMITTED.

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. What are different transaction levels in SQL?**

**Q. What are the different locks in SQL?**

*ToDo*

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**# 18. SQL Functions**

**Q. What is a function in SQL Server?**

**Q. What are the different types of functions in SQL Server?**

**Q. What are the reporting aggregate functions available in SQL?**

In database management, an aggregate function is a function where the values of multiples rows are grouped to form a single value.

| **Sl.No** | **Function** | **Description** |
| --- | --- | --- |
| 01. | COUNT | Return the number of rows in a certain table/view |
| 02. | SUM | Accumulate the values |
| 03. | AVG | Returns the average for a group of values |
| 04. | MIN | Returns the smallest value of the group |
| 05. | MAX | Returns the largest value of the group |

**Q. What are aggregate and scalar functions?**

**Q. What are all the Common SQL Function?**

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**# 19. SQL View**

**Q. What is View in SQL?**

A view is a virtual table that is a result of a query. They can be extremely useful and are often used as a security mechanism, letting users access the data through the view, rather than letting them access the underlying base table:

**Syntax:**

CREATE VIEW view\_name AS

SELECT column1, column2, ...

FROM table\_name

WHERE condition;

**Example:**

--- Creating a View

CREATE VIEW trackView AS

SELECT id, album\_id, title, track\_number, duration DIV 60 AS m, duration MOD 60 AS s

FROM track;

SELECT \* FROM trackView;

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**# 20. SQL Triggers**

**Q. What are the triggers in SQL?**

A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

**Syntax:**

CREATE [OR REPLACE ] TRIGGER <trigger\_name>

{BEFORE | AFTER | INSTEAD OF }

{INSERT [OR] | UPDATE [OR] | DELETE}

ON <table\_name>

[FOR EACH ROW]

WHEN (condition)

[trigger\_body]

**Example - 01:**

CREATE TRIGGER employee\_name

after INSERT

on

employee

for each row

BEGIN

UPDATE employee set full\_name = first\_name || ' ' || last\_name;

END;

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. Why and when to use a trigger?**

**Q. What are the different types of triggers?**

**Q. How many TRIGGERS are possible in MySql?**

*ToDo*

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**# 21. SQL Cursors**

**Q. What is a cursor?**

When we execute any SQL operations, SQL Server opens a work area in memory which is called Cursor. When it is required to perform the row by row operations which are not possible with the set-based operations then cursor is used.

There are two of cursors:

**1. Implicit Cursor:**

Implicit Cursors are also known as Default Cursors of SQL SERVER. These Cursors are allocated by SQL SERVER when the user performs DML operations.

**2. Explicit Cursor:**

* When the programmer wants to perform the row by row operations for the result set containing more than one row, then he explicitly declare a cursor with a name.
* They are managed by OPEN, FETCH and CLOSE.
* %FOUND, %NOFOUND, %ROWCOUNT and %ISOPEN attributes are used in both types of cursors.

**1. Declare Cursor Object:**

**Syntax:**

--- DECLARE cursor\_name CURSOR FOR SELECT \* FROM table\_name

DECLARE s1 CURSOR FOR SELECT \* FROM studDetails

**2. Open Cursor Connection:**

-- OPEN cursor\_connection

OPEN s1

**3. Fetch Data from cursor:**

There are total 6 methods to access data from cursor.

* **FIRST** - is used to fetch only the first row from cursor table.
* **LAST** - is used to fetch only last row from cursor table.
* **NEXT** - is used to fetch data in forward direction from cursor table.
* **PRIOR** - is used to fetch data in backward direction from cursor table.
* **ABSOLUTE** - n is used to fetch the exact nth row from cursor table.
* **RELATIVE** - n is used to fetch the data in incremental way as well as decremental way.

FETCH FIRST FROM s1

FETCH LAST FROM s1

FETCH NEXT FROM s1

FETCH PRIOR FROM s1

FETCH ABSOLUTE 7 FROM s1

FETCH RELATIVE -2 FROM s1

**4. Close cursor connection:**

--- CLOSE cursor\_name

CLOSE s1

**5. Deallocate cursor memory:**

--- DEALLOCATE cursor\_name

DEALLOCATE s1

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. When is the Explicit Cursor Used?**

**# 22. SQL Stored Procedures**

**Q. Why stored procedures are called as executable code?**

Stored procedure stored inside the database. This also includes the executable code that usually collects and customizes the operations like insert, encapsulation, etc. These stored procedures are used as APIs for simplicity and security purposes. The implementation of it allows the developers to have procedural extensions to the standard SQL syntax. Stored procedure doesn'\t come as a part of relational database model, but can be included in many implementations commercially.

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. What are the advantages of using Stored Procedures?**

* Procedure can reduce network traffic and latency, and can enhance application performance.
* Procedure execution plans can be reused, staying cached in the management tool's memory, reducing its overhead.
* Procedures provide the benefit of code reuse.
* The logic can be encapsulated using procedures and can help to change procedure's code without interacting to application.
* Procedures give more security to our data.

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. What is stored procedure in SQL?**

Stored Procedures are created to perform one or more DML operations on Database. It is nothing but the group of SQL statements that accepts some input in the form of parameters and performs some task and may or may not returns a value.

**Syntax:**

CREATE or REPLACE PROCEDURE name(parameters)

IS

variables;

BEGIN

//statements;

END;

**Example:**

CREATE PROCEDURE SelectAllCustomers

AS

SELECT \* FROM Customers

GO;

Execute the stored procedure above as follows:

EXEC SelectAllCustomers;

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. What is Stored Routine in SQL?**

A stored routine is a set of SQL statements that are stored on the database server and can be used by any client with permission to use them. This provides a number of benefits.

1. Database operations are normalized so various applications will operate uniformly, even when written in different languages and operating on different platforms.
2. Stored routines are easy to maintain, because they're all in one place rather than distributed among different applications.
3. Traffic is reduced between the client and server, because data is processed on the server.
4. security is enhanced by allowing clients to run with reduced permissions while still being able to perform necessary database operations.

There are two different kinds of stored routines.

a) Stored functions return a value, and are used in the context of an expression. b) Stored procedures are called separately, using the call statement, and may return result sets or set variables.

**Example 01:** Stored Functions

DROP FUNCTION IF EXISTS track\_len;

CREATE FUNCTION track\_len(seconds INT)

RETURNS VARCHAR(16) DETERMINISTIC

RETURN CONCAT\_WS(':', seconds DIV 60, LPAD(seconds MOD 60, 2, '0' ));

SELECT title, track\_len(duration) FROM track;

SELECT a.artist AS artist,

a.title AS album,

t.title AS track,

t.track\_number AS trackno,

track\_len(t.duration) AS length

FROM track AS t

JOIN album AS a

ON a.id = t.album\_id

ORDER BY artist, album, trackno

;

**Example 02:** Stored Procedures

DROP PROCEDURE IF EXISTS list\_albums;

DELIMITER //

CREATE PROCEDURE list\_albums ()

BEGIN

SELECT \* FROM album;

END

//

DELIMITER ;

CALL list\_albums();

**Example 03:** Stored Procedures with parameter

DROP PROCEDURE IF EXISTS list\_albums;

DELIMITER //

CREATE PROCEDURE list\_albums (a VARCHAR(255))

BEGIN

SELECT a.artist AS artist,

a.title AS album,

t.title AS track,

t.track\_number AS trackno,

track\_len(t.duration) AS length

FROM track AS t

JOIN album AS a

ON a.id = t.album\_id

WHERE a.artist LIKE a

ORDER BY artist, album, trackno

;

END //

DELIMITER ;

CALL list\_albums('%hendrix%');

**Example 04:** Drop Stored Procedures & Stored Functions

DROP FUNCTION IF EXISTS track\_len;

DROP PROCEDURE IF EXISTS total\_duration;

**Q. What is the difference between Stored Procedure and User Defined Function?**

**Q. How can you raise custom errors from stored procedure?**

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**# 23. Miscellaneous**

**Q. How do you find third highest salary**

SELECT \* from employees order by salary limit 2,1;

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. What is the STUFF function and how does it differ from the REPLACE function?**

* Using STUFF function we can overwrite the specified characters of a string. The syntax of STUFF function is: STUFF (stringToChange, startIndex, length, new\_characters )

where stringToChange is the string which will have the characters those we want to overwrite, startIndex is the starting position, length is the number of characters in the string that are to be overwrited, and new\_characters are the new characters to write into the string.

* While REPLACE function is used to replace specified character at all its existing occurrences.
* The syntax of REPLACE function is REPLACE (string\_to\_change, string\_to\_Replace, new\_tring).
* Every occurrence of string\_to\_change will be replaced by new\_string.

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. What is RANK function?**

* RANK function can be used to give a rank to each row returned from a SELECT statment.
* For using this function first specify the function name, followed by the empty parentheses.
* Then mention the OVER function. For this function, you have to pass an ORDER BY clause as an argument. The clause identifies the column on which you are going to apply the RANK function.

For Example: SELECT RANK() OVER(ORDER BY BirthDate DESC) AS [RowNumber], FirstName, BirthDate FROM EmpDetails

* In the result you will see that the eldest employee got the first rank and the youngest employee got the last rank. Here the rows with equal age will get same ranks.
* The rank depends on the row's position in the result set, but not on the sequential number of the row.

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. What are the reasons of poor performance of query?**

Following are the reasons for the poor performance of a query:

* No indexes.
* Excess recompilations of stored procedures.
* Procedures and triggers without SET NOCOUNT ON.
* Poorly written query with unnecessarily complicated joins.
* Highly normalized database design.
* Excess usage of cursors and temporary tables.
* Queries with predicates that use comparison operators between different columns of the same table.
* Queries with predicates that use operators, and any one of the following are true:
* There are no statistics on the columns involved on either side of the operators.
* The distribution of values in the statistics is not uniform, but the query seeks a highly selective value set. This situation can be especially true if the operator is anything other than the equality (=) operator.
* The predicate uses the not equal to (!=) comparison operator or the NOT logical operator.
* Queries that use any of the SQL Server built-in functions or a scalar-valued, user-defined function whose argument is not a constant value.
* Queries that involve joining columns through arithmetic or string concatenation operators.
* Queries that compare variables whose values are not known when the query is compiled and optimized.

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. What are the MySQL Engines?**

**1. InnoDB:**

The default storage engine in MySQL 8.0. InnoDB is a transaction-safe (ACID compliant) storage engine for MySQL that has commit, rollback, and crash-recovery capabilities to protect user data. InnoDB row-level locking (without escalation to coarser granularity locks) and Oracle-style consistent nonlocking reads increase multi-user concurrency and performance. InnoDB stores user data in clustered indexes to reduce I/O for common queries based on primary keys.

To maintain data integrity, InnoDB also supports FOREIGN KEY referential-integrity constraints.

**2. MyISAM:**

These tables have a small footprint. Table-level locking limits the performance in read/write workloads, so it is often used in read-only or read-mostly workloads in Web and data warehousing configurations.

**3. Memory:**

Stores all data in RAM, for fast access in environments that require quick lookups of non-critical data. This engine was formerly known as the HEAP engine. Its use cases are decreasing; InnoDB with its buffer pool memory area provides a general-purpose and durable way to keep most or all data in memory, and NDBCLUSTER provides fast key-value lookups for huge distributed data sets.

**4. CSV:**

Its tables are really text files with comma-separated values. CSV tables let you import or dump data in CSV format, to exchange data with scripts and applications that read and write that same format. Because CSV tables are not indexed, you typically keep the data in InnoDB tables during normal operation, and only use CSV tables during the import or export stage.

**5. Archive:**

These compact, unindexed tables are intended for storing and retrieving large amounts of seldom-referenced historical, archived, or security audit information.

**6. Blackhole:**

The Blackhole storage engine accepts but does not store data, similar to the Unix /dev/null device. Queries always return an empty set. These tables can be used in replication configurations where DML statements are sent to slave servers, but the master server does not keep its own copy of the data.

**7. NDB:**

This clustered database engine is particularly suited for applications that require the highest possible degree of uptime and availability.

**8. Merge:**

Enables a MySQL DBA or developer to logically group a series of identical MyISAM tables and reference them as one object. Good for VLDB environments such as data warehousing.

**9. Federated:**

Offers the ability to link separate MySQL servers to create one logical database from many physical servers. Very good for distributed or data mart environments.

**10. Example:**

This engine serves as an example in the MySQL source code that illustrates how to begin writing new storage engines. It is primarily of interest to developers. The storage engine is a “stub” that does nothing. You can create tables with this engine, but no data can be stored in them or retrieved from them.

[**↥ back to top**](https://github.com/learning-zone/sql-basics#table-of-contents)

**Q. How to find the unique values if the value in the column is repeated?**

**Q. How to test performance of database?**

**Q. What is SQL Profiler?**

**Q. How to get @@ERROR and @@ROWCOUNT at the same time?**

**Q. Explain about buffer cash and log Cache in SQL Server?**

### Schema

employees

* employee\_id
* department\_id
* boss\_id
* name
* salary

departments

* department\_id
* name

### Questions

1. List employees (names) who have a bigger salary than their boss
2. List employees who have the biggest salary in their departments
3. List departments that have less than 3 people in it
4. List all departments along with the number of people there (tricky - people often do an "inner join" leaving out empty departments)
5. List employees that don't have a boss in the same department
6. List all departments along with the total salary there

[Raw](https://gist.github.com/mjhea0/5667162/raw/b7a99fe2ad5ca7116d2a482667a0dfdaf5264cf5/2%2520-%2520create_schema.sql)

[**2 - create\_schema.sql**](https://gist.github.com/mjhea0/5667162#file-2-create_schema-sql)

|  |  |
| --- | --- |
|  | CREATE TABLE `employees` ( |
|  | `employee\_id` int(11) unsigned NOT NULL AUTO\_INCREMENT, |
|  | `department\_id` int(11) DEFAULT NULL, |
|  | `boss\_id` int(11) DEFAULT NULL, |
|  | `name` varchar(255) DEFAULT NULL, |
|  | `salary` varchar(255) DEFAULT NULL, |
|  | PRIMARY KEY (`employee\_id`) |
|  | ); |
|  |  |
|  | CREATE TABLE `departments` ( |
|  | `department\_id` int(11) unsigned NOT NULL AUTO\_INCREMENT, |
|  | `name` varchar(255) DEFAULT NULL, |
|  | PRIMARY KEY (`department\_id`) |
|  | ); |

[Raw](https://gist.github.com/mjhea0/5667162/raw/b7a99fe2ad5ca7116d2a482667a0dfdaf5264cf5/3%2520-%2520insert_data.sql)

[**3 - insert\_data.sql**](https://gist.github.com/mjhea0/5667162#file-3-insert_data-sql)

|  |  |
| --- | --- |
|  | INSERT INTO `employees` (`employee\_id`, `department\_id`, `boss\_id`, `name`, `salary`) |
|  | VALUES |
|  | (1,1,0,'manager','80000'), |
|  | (2,1,1,'emp1','60000'), |
|  | (3,1,1,'emp2','50000'), |
|  | (4,1,1,'emp3','95000'), |
|  | (5,1,1,'emp4','75000'); |
|  |  |
|  | INSERT INTO `departments` (`department\_id`, `name`) |
|  | VALUES |
|  | (1,'IT'), |
|  | (2,'HR'), |
|  | (3,'Sales'), |
|  | (4,'Marketing'); |

[Raw](https://gist.github.com/mjhea0/5667162/raw/b7a99fe2ad5ca7116d2a482667a0dfdaf5264cf5/4%2520-%2520answers.md)

[**4 - answers.md**](https://gist.github.com/mjhea0/5667162#file-4-answers-md)

## Answers

### List employees (names) who have a bigger salary than their boss

SELECT e.name AS 'Employee Name', e2.name AS 'Boss',

e.salary AS 'Employee salary', e2.salary AS 'Boss salary'

FROM employees e

JOIN employees e2 ON e.boss\_id = e2.employee\_id

WHERE e2.salary < e.salary;

### List employees who have the biggest salary IN their departments

SELECT e.name AS 'Employee Name', d.Name AS 'Department', e.Salary AS 'Salary'

FROM departments d JOIN employees e

ON e.department\_id = d.department\_id

GROUP BY d.department\_id

ORDER BY e.salary DESC

### List departments that have less than 3 people IN it

SELECT d.Name AS 'Department'

FROM departments d JOIN employees e

ON e.department\_id = d.Department\_id

GROUP BY d.department\_id

HAVING COUNT(e.employee\_id) < 3

### List ALL departments along WITH the NUMBER OF people there (tricky - people often do an "inner join" leaving OUT empty departments)

SELECT d.name AS 'Department', COUNT(e.employee\_id) AS '# of Employees'

FROM departments d LEFT OUTER JOIN employees e

ON e.department\_id = d.department\_id

GROUP BY d.Department\_id

### List employees that don't have a boss in the same department

SELECT e.name AS 'Employee Name', e2.name AS 'Boss', d.name AS "Employee's Department", d2.name AS "Boss' Department"

FROM employees e

JOIN employees e2 ON e.boss\_id = e2.employee\_id

JOIN departments d ON e.department\_id = d.department\_id

JOIN departments d2 ON e2.department\_id = d2.department\_id

WHERE e.boss\_id != 0

AND d.department\_id != d2.department\_id

### List all departments along with the total salary there

SELECT d.name AS 'Department', SUM(e.salary) AS 'Total Salary'

FROM departments d LEFT OUTER JOIN employees e

ON d.department\_id = e.department\_id

GROUP BY d.department\_id

[](https://gist.github.com/chromano)

### [chromano](https://gist.github.com/chromano)commented [on Jun 2, 2013](https://gist.github.com/mjhea0/5667162?permalink_comment_id=838830#gistcomment-838830)

In #2, it is supposed to return one entry per department as I understood it.

Top of Form

Bottom of Form

[](https://gist.github.com/chriscrowe)

### [chriscrowe](https://gist.github.com/chriscrowe)commented [on Jun 2, 2013](https://gist.github.com/mjhea0/5667162?permalink_comment_id=839084#gistcomment-839084)

I think chromano is correct. Here was my response:

SELECT d.name as department, e.name as max\_earner, MAX(e.salary) as salary

FROM employees e

JOIN departments d ON e.department\_id=d.id

GROUP BY e.department\_id;

Note that my solutions use a simple id column as a pk for each table instead of employee\_id and department\_id.

Top of Form

Bottom of Form

[](https://gist.github.com/chriscrowe)

### [chriscrowe](https://gist.github.com/chriscrowe)commented [on Jun 3, 2013](https://gist.github.com/mjhea0/5667162?permalink_comment_id=839092#gistcomment-839092)

Also #4 -- your solution will show counts of 1 for empty departments. Here is a working solution that shows zeros:

SELECT d.name, COUNT(e.department\_id) AS num\_employees

FROM departments d LEFT JOIN employees e ON d.id=e.department\_id

GROUP BY d.id

Top of Form

Bottom of Form

[](https://gist.github.com/chriscrowe)

### [chriscrowe](https://gist.github.com/chriscrowe)commented [on Jun 3, 2013](https://gist.github.com/mjhea0/5667162?permalink_comment_id=839096#gistcomment-839096)

#5 is unnecessarily complex.

SELECT e.name

FROM employees e JOIN employees b ON e.boss\_id=b.id

WHERE e.department\_id != b.department\_id

OR b.boss\_id is NULL

Top of Form

Bottom of Form

[](https://gist.github.com/kalypzo)

### [kalypzo](https://gist.github.com/kalypzo)commented [on Aug 5, 2013](https://gist.github.com/mjhea0/5667162?permalink_comment_id=880171#gistcomment-880171)

The solutions given for problem #2 contain data inconsistencies. The correct answer is: IT,emp3,95000

In the original posted answer, the query will return the wrong salary and wrong employee. The group by is executed before the order by clause and returns the first row that matches department\_id. As a result, the salary is not really the greatest per department. Returns: IT,manager,80000. incorrect

In skandocious answer, this query will fail completely for other databases like MS SQL and Oracle, because all non-aggregated columns must be specified in the group by clause for consistency purposes. The correct salary is returned, but with the wrong associated employee. The max(salary) returns the max salary per department ID; however, since we didn't specify the column names in the group by clause, the data is no longer consistent. We now return the first possible employee name for the matching department courtesy of the join clause, but with the maximum salary per department. Returns: IT, manager, 95000. incorrect

Below are two possible solutions:

First solution - Returns only one person for each department with the highest salary:

SELECT \*

FROM ( SELECT dept.Name Department, emp.Name Employee, emp.Salary Salary

FROM Departments dept

JOIN Employees emp ON emp.department\_id = dept.department\_id

ORDER BY salary desc ) result

GROUP BY Department;

We first get the departments, employees and their relative salaries, and organize salaries in descending order. The outer query uses a group by, grabbing the first row of each record associated with their department. Since the first result set is ordered by salary in descending order, the outer group by will choose the first record, which will be the highest salary. One issue with this solution is we will not return each employee that shares the highest salary.

Second solution - Returns one or more people for each department with the highest salary:

SELECT result.Name Department, emp2.Name Employee, result.salary Salary

FROM ( SELECT dept.name, dept.department\_id, max(emp1.salary) salary

FROM Departments dept

JOIN Employees emp1 ON emp1.department\_id = dept.department\_id

GROUP BY dept.name, dept.department\_id ) result

JOIN Employees emp2 ON emp2.department\_id = result.department\_id

WHERE emp2.salary = result.salary;

This one will return each employee that shares the highest salary.

Top of Form

Bottom of Form

[](https://gist.github.com/kushalbhola)

### [kushalbhola](https://gist.github.com/kushalbhola)commented [on Nov 3, 2018](https://gist.github.com/mjhea0/5667162?permalink_comment_id=2750350#gistcomment-2750350)

Solution to the 5th can be better written as follows  
select e2.\*, e1.department\_ID from #employees e1 inner join #employees e2 on e1.employee\_id = e2.boss\_id where e1.department\_ID <> e2.department\_ID

Top of Form

Bottom of Form

[](https://gist.github.com/GreyHatt)

### [GreyHatt](https://gist.github.com/GreyHatt)commented [on Jun 13, 2019](https://gist.github.com/mjhea0/5667162?permalink_comment_id=2942544#gistcomment-2942544) •

### edited

Solution to the 2nd is wrong.  
kalypzo your query is fine but it gives the department not the employee  
Question asked to find out the employee not the department. Therefore why do we need to join department table in first place.  
Please comment if i'm wrong.  
correct solution:  
Select A.Employee\_id,A.name from (Select employee\_id,name,DENSE\_RANK() OVER (PARTITION BY department\_id ORDER BY Salary DESC) AS Rnk from Employees) A where A.Rnk=1

Top of Form

Bottom of Form

[](https://gist.github.com/moe45673)

### [moe45673](https://gist.github.com/moe45673)commented [on Jul 4, 2022](https://gist.github.com/mjhea0/5667162?permalink_comment_id=4220414#gistcomment-4220414)

Thank you so much for the Table Creation/Populating code. I want to practise these questions but do not want the answers. I could have whipped them up myself but.... this makes things so much more streamlined. thank you!