**SQL Concepts**

Contents

[**Important SQL concepts** 2](#_Toc40897537)

[**SQL questions** 3](#_Toc40897538)

[**Comparison of SQL/MySQL and Oracle** 4](#_Toc40897539)

# **Important SQL concepts**

|  |  |
| --- | --- |
| **Differences** |  |
| UNION vs UNIONALL | **UNION** filters and displays unique result sets. **UNIONALL** displays duplicates |
| DROP vs TRUNCATE vs DELETE | The **DROP** command removes a table from the database. All the tables' rows, indexes and privileges will also be removed. No DML triggers will be fired. The operation cannot be rolled back.  **TRUNCATE** removes all rows from a table. The operation cannot be rolled back and no triggers will be fired. As such, TRUNCATE is faster and doesn't use as much undo space as a DELETE. Table level lock will be added when Truncating.  The **DELETE** command is used to remove rows from a table. A WHERE clause can be used to only remove some rows. If no WHERE condition is specified, all rows will be removed. After performing a DELETE operation you need to COMMIT or ROLLBACK the transaction to make the change permanent or to undo it. Note that this operation will cause all DELETE triggers on the table to fire. Row level lock will be added when deleting. |
| Primary Key vs Unique Key | <https://www.essentialsql.com/primary-and-unique-key/>  The main purpose of the primary key is to provide a means to identify each record in the table. The primary key provides a means to identity the row, using data within the row. There can only be one primary key for a table.  PK consists of one or more columns. PK enforces the entity integrity of the table. All columns defined must be defined as NOT NULL.  A unique key is also called a unique constraint. A unique constraint can be used to ensure rows are unique within the database. There can be multiple unique keys defined on a table. Unique Keys result in NONCLUSTERED Unique Indexes by default. One or more columns make up a unique key. Column may be NULL, but on one NULL per column is allowed. |
| Clustered Index vs Non-Clustered Index | **Clustered Index**- Only one per table. Faster to read than non clustered as data is physically stored in index order. Do not need extra space to store logical structure.  **Non Clustered Index** - Can be used many times per table. Quicker for insert and update operations than a clustered index. Both types of index will improve performance when select data with fields that use the index but will slow down update and insert operations. Creates a logical order for data rows and use pointers to physical data files  Because of the slower insert and update clustered indexes should be set on a field that is normally incremental ie Id or Timestamp.  SQL Server will normally only use an index if its selectivity is above 95%. |
| Normalized vs Denormalized database | Normalized tables : Like data is organized into one table and other related data is put into a different table. You get to each piece of data through relationships to each table, mainly join tables. The good thing is, normalization reduces redundancy and maintains data integrity. But, much like the downside of Rails, normalized databases can cause queries to slow down, especially when dealing with a shit ton (technical term) of data.  Denormalized tables : Denormalization is a database optimization technique in which we add redundant data to one or more tables. This can help us avoid costly joins in a relational database. Note that denormalization does not mean not doing normalization. It is an optimization technique that is applied after doing normalization. Under denormalization, we decide that we’re okay with some redundancy and some extra effort to update the database in order to get the efficiency advantages of fewer joins.  **Pros of Denormalization:-**  Retrieving data is faster since we do fewer joins  Queries to retrieve can be simpler(and therefore less likely to have bugs),  since we need to look at fewer tables.  **Cons of Denormalization:-**  Updates and inserts are more expensive.  Denormalization can make update and insert code harder to write.  Data may be inconsistent . Which is the “correct” value for a piece of data?  Data redundancy necessitates more storage. |

# **SQL questions**

* What criteria is used for creating Index in a table

In general, you should create an index on a column in any of the following situations:

1. The column is queried frequently.
2. A referential integrity constraint exists on the column.
3. A UNIQUE key integrity constraint exists on the column.

You can create an index on any column; however, if the column is not used in any of these situations, creating an index on the column does not increase performance and the index takes up resources unnecessarily.

## **Comparison of SQL/MySQL and Oracle**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **SQL** | **MySQL** | **Oracle** |
| Fetch top number of records which match a criteria | Select Top number \* from table\_name where [condition] | Select \* from table\_name where [condition] limit <number> | --used to limit the number of records returned by a query  Select \* from table\_name where rownum<[number] |