Database concepts and interview questions

What is a Database?

A database is an organized collection of structured information, or data, typically stored electronically in a computer system. A database is usually controlled by a database management system (DBMS). Most databases use structured query language (SQL) for writing and querying data.

What are Anomalies in db?

Anomalies are problems that can occur in poorly planned, unnormalized databases where all the data is stored in one table (a flat-file database). Insertion Anomaly - The nature of a database may be such that it is not possible to add a required piece of data unless another piece of unavailable data is also added.

The SQL CASE Statement

The CASE statement goes through conditions and returns a value when the first condition is met (like an IF-THEN-ELSE statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.

If there is no ELSE part and no conditions are true, it returns NULL.

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    WHEN conditionN THEN resultN
    ELSE result
END;
```

```
SELECT OrderID, Quantity,
CASE
    WHEN Quantity > 30 THEN 'The quantity is greater than 30'
    WHEN Quantity = 30 THEN 'The quantity is 30'
    ELSE 'The quantity is under 30'
END AS QuantityText
FROM OrderDetails;
```

SQL JOIN

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.
Let's look at a selection from the "Orders" table:

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

Then, look at a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

Notice that the "CustomerID" column in the "Orders" table refers to the "CustomerID" in the "Customers" table. The relationship between the two tables above is the "CustomerID" column.
Then, we can create the following SQL statement (that contains an INNER JOIN), that selects records that have matching values in both tables:

Example

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

Or

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders, Customers
Where Orders.CustomerID=Customers.CustomerID;
```

and it will produce something like this:

OrderID	CustomerName	OrderDate
10308	Ana Trujillo Emparedados y helados	9/18/1996

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

- **(INNER) JOIN:** Returns records that have matching values in both tables
-

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders, Customers
Where Orders.CustomerID=Customers.CustomerID;
```

- **LEFT (OUTER) JOIN:** Returns all records from the left table, and the matched records from the right table

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders, Customers
Where Orders.CustomerID=Customers.CustomerID(+);
```

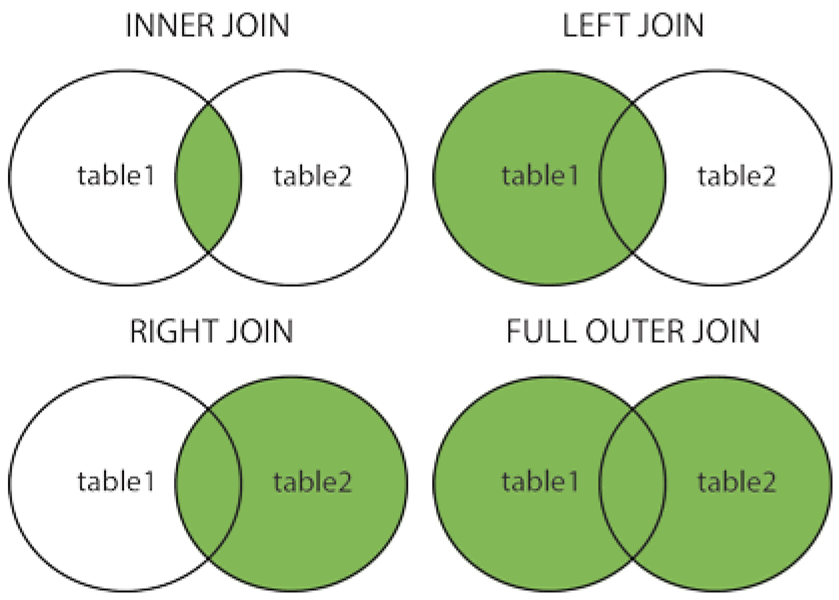
- RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders, Customers
Where Orders.CustomerID(+)=Customers.CustomerID;
```

- FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders, Customers
```

- SELF JOIN: A join in which a table is joined with itself (which is also called Unary relationships).



Normalization in database

Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy(repetition) and undesirable characteristics like Insertion, Update and Deletion Anomalies. It is a multi-step process that puts data into tabular form, removing duplicated data from the relation tables.

Normalization is used for mainly two purposes,

- Eliminating redundant(useless) data.
- Ensuring data dependencies make sense i.e data is logically stored.

Normalization Rule

Normalization rules are divided into the following normal forms:

1. First Normal Form

2. Second Normal Form
3. Third Normal Form
4. BCNF
5. Fourth Normal Form

First Normal Form (1NF)

For a table to be in the First Normal Form, it should follow the following 4 rules:

1. It should only have single(atomic) valued attributes/columns.
2. Values stored in a column should be of the same domain.
3. All the columns in a table should have unique names.
4. And the order in which data is stored, does not matter.

Second Normal Form (2NF)

For a table to be in the Second Normal Form,

1. It should be in the First Normal form.
2. And, it should not have Partial Dependency.

What is **Partial Dependency**? Do not worry about it. First let's understand what is **Dependency** in a table?

What is Dependency?

Let's take an example of a **Student** table with columns student_id, name, reg_no(registration number), branch and address(student's home address).

student_id	name	reg_no	branch	address

In this table, student_id is the primary key and will be unique for every row, hence we can use student_id to fetch any row of data from this table

Even for a case, where student names are the same, if we know the student_id we can easily fetch the correct record.

student_id	name	reg_no	branch	address
10	Akon	07-WY	CSE	Kerala
11	Akon	08-WY	IT	Gujarat

Hence we can say a **Primary Key** for a table is the column or a group of columns(composite key) which can uniquely identify each record in the table.

I can ask for the branch name of a student with student_id **10**, and I can get it. Similarly, if I ask for the name of a student with student_id **10** or **11**, I will get it. So all I need is student_id and every other column **depends** on it, or can be fetched using it.

This is Dependency and we also call it **Functional Dependency**.

What is Partial Dependency?

Now that we know what dependency is, we are in a better state to understand what partial dependency is.

For a simple table like Student, a single column like student_id can uniquely identify all the records in a table.

But this is not true all the time. So now let's extend our example to see if more than 1 column together can act as a primary key.

Let's create another table for **Subject**, which will have subject_id and subject_name fields and subject_id will be the primary key.

subject_id	subject_name
1	Java
2	C++
3	Php

Now we have a **Student** table with student information and another table **Subject** for storing subject information.

Let's create another table **Score**, to store the **marks** obtained by students in the respective subjects. We will also be saving the name **of the teacher** who teaches that subject along with marks.

score_id	student_id	subject_id	marks	teacher
1	10	1	70	Java Teacher
2	10	2	75	C++ Teacher
3	11	1	80	Java Teacher

In the score table we are saving the **student_id** to know which student's marks are these and **subject_id** to know for which subject the marks are for.

Together, student_id + subject_id forms a **Candidate Key**(learn about Database Keys) for this table, which can be the **Primary key**.

Confused, How can this combination be a primary key?

See, if I ask you to get me marks of a student with student_id 10, can you get it from this table? No, because you don't know for which subject. And if I give you subject_id, you would not know for which student. Hence we need student_id + subject_id to uniquely identify any row.

But where is Partial Dependency?
Now if you look at the **Score** table, we have a column names teacher which is only dependent on the subject, for Java it's Java Teacher and for C++ it's C++ Teacher & so on.

Now as we just discussed that the primary key for this table is a composition of two columns which is student_id & subject_id but the teacher's name only depends on subject, hence the subject_id, and has nothing to do with student_id.

This is **Partial Dependency**, where an attribute in a table depends on only a part of the primary key and not on the whole key.

How to remove Partial Dependency?

There can be many different solutions for this, but our objective is to remove the teacher's name from the Score table.

The simplest solution is to remove the column's teacher from the Score table and add it to the Subject table. Hence, the Subject table will become:

subject_id	subject_name	teacher
1	Java	Java Teacher
2	C++	C++ Teacher

3	Php	Php Teacher

And our Score table is now in the second normal form, with no partial dependency.

score_id	student_id	subject_id	marks
1	10	1	70
2	10	2	75
3	11	1	80

Third Normal Form (3NF)

A table is said to be in the Third Normal Form when,

1. It is in the Second Normal form.
2. And, it doesn't have Transitive Dependency.

So let's use the same example, where we have 3 tables, **Student**, **Subject** and **Score**.

Student Table

student_id	name	reg_no	branch	address
10	Akon	07-WY	CSE	Kerala
11	Akon	08-WY	IT	Gujarat

12	Bkon	09-WY	IT	Rajasthan

Subject Table

subject_id	subject_name	teacher
1	Java	Java Teacher
2	C++	C++ Teacher
3	Php	Php Teacher

Score Table

score_id	student_id	subject_id	marks
1	10	1	70
2	10	2	75
3	11	1	80

In the Score table, we need to store some more information, which is the exam name and total marks, so let's add 2 more columns to the Score table.

score_id	student_id	subject_id	marks	exam_name	total_marks

Requirements for Third Normal Form

For a table to be in the third normal form,

1. It should be in the Second Normal form.
2. And it should not have Transitive Dependency.

What is Transitive Dependency?

With exam_name and total_marks added to our Score table, it saves more data now. Primary key for our Score table is a composite key, which means it's made up of two attributes or columns → **student_id + subject_id**.

Our new column exam_name depends on both student and subject. For example, a mechanical engineering student will have a Workshop exam but a computer science student won't. And for some subjects you have Practical exams and for some you don't. So we can say that exam_name is dependent on both student_id and subject_id.

And what about our second new column total_marks? Does it depend on our Score table's primary key?

Well, the column total_marks depends on exam_name as with exam type the total score changes. For example, practicals are of less marks while theory exams are of more marks.

But, exam_name is just another column in the score table. It is not a primary key or even a part of the primary key, and total_marks depends on it.

This is **Transitive Dependency**. When a non-prime attribute depends on other non-prime attributes rather than depending upon the prime attributes or primary key.

How to remove Transitive Dependency?

Again the solution is very simple. Take out the columns exam_name and total_marks from Score table and put them in an **Exam** table and use the exam_id wherever required.

Score Table: In 3rd Normal Form

score_id	**student_id**	**subject_id**	**marks**	**exam_id**

The new Exam table

exam_id	**exam_name**	**total_marks**
1	Workshop	200
2	Mains	70
3	Practicals	30

Advantage of removing Transitive Dependency

The advantage of removing transitive dependency is,

- Amount of data duplication is reduced.
- Data integrity achieved.

Boyce and Codd Normal Form (BCNF)

Boyce and Codd Normal Form is a higher version of the Third Normal form. This form deals with a certain type of anomaly that is not handled by 3NF. A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF. For a table to be in BCNF, following conditions must be satisfied:

- R must be in 3rd Normal Form
- and, for each functional dependency (X → Y), X should be a super Key.

For a table to satisfy the Boyce-Codd Normal Form, it should satisfy the following two conditions:

1. It should be in the **Third Normal Form**.
2. And, for any dependency A → B, A should be a **super key**.

The second point sounds a bit tricky, right? In simple words, it means that for a dependency A → B, A cannot be a **non-prime attribute**, if B is a **prime attribute**.

Time for an Example

Below we have a college enrolment table with columns student_id, subject and professor.

student_id	**subject**	**professor**
101	Java	P.Java
101	C++	P.Cpp
102	Java	P.Java2
103	C#	P.Cash
104	Java	P.Java

As you can see, we have also added some sample data to the table.

In the table above:

- One student can enrol for multiple subjects. For example, student with **student_id** 101, has opted for subjects - Java & C++
- For each subject, a professor is assigned to the student.
- And, there can be multiple professors teaching one subject like we have for Java.

What do you think should be the **Primary Key**?

Well, in the table above student_id, subject together form the primary key, because using student_id and subject, we can find all the columns of the table.

One more important point to note here is, one professor teaches only one subject, but one subject may have two different professors.

Hence, there is a dependency between subject and professor here, where subject depends on the professor's name.

This table satisfies the **1st Normal form** because all the values are atomic, column names are unique and all the values stored in a particular column are of the same domain.

This table also satisfies the **2nd Normal Form** as there is no **Partial Dependency**.

And, there is no **Transitive Dependency**, hence the table also satisfies the **3rd Normal Form**.

But this table is not in **Boyce-Codd Normal Form**.

Why is this table not in BCNF?

In the table above, student_id, subject form primary key, which means subject column is a prime attribute.

But, there is one more dependency, professor → subject.

And while subject is a prime attribute, professor is a **non-prime attribute**, which is not allowed by BCNF.

How to satisfy BCNF?

To make this relation(table) satisfy BCNF, we will decompose this table into two tables,

student table and **professor** table.

Below we have the structure for both the tables.

Student Table

student_id	**p_id**
101	1
101	2
and so on...	

And, **Professor Table**

p_id	**professor**	**subject**
1	P.Java	Java
2	P.Cpp	C++
and so on...		

And now, this relation satisfy Boyce-Codd Normal Form. In the next tutorial we will learn about the **Fourth Normal Form**.

Difference between Normalization and Denormalization

Sr. No.	Key	Normalization	Denormalization
1	Implementation	Normalization is used to remove redundant data from the database and to store non-redundant and consistent data into it.	Denormalization is used to combine multiple table data into one so that it can be queried quickly.
2	Focus	Normalization mainly focuses on clearing the database from unused data and to reduce the data redundancy and inconsistency.	Denormalization on the other hand focuses on to achieve the faster execution of the queries through introducing redundancy.

3	Number of Tables	During Normalization as data is reduced so a number of tables are deleted from the database hence tables are lesser in number.	On another hand during Denormalization data is integrated into the same database and hence a number of tables to store that data increases in number.
4	Memory consumption	Normalization uses optimized memory and hence faster in performance.	On the other hand, Denormalization introduces some sort of wastage of memory.
5	Data integrity	Normalization maintains data integrity i.e. any addition or deletion of data from the table will not create any mismatch in the relationship of the tables.	Denormalization does not maintain any data integrity.
6	Where to use	Normalization is generally used where number of insert/update/delete operations are performed and joins of those tables are not expensive.	On the other hand Denormalization is used where joins are expensive and frequent query is executed on the table

Distinct vs unique

The SELECT DISTINCT statement is used to return only distinct (different) values. Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

Unique is a keyword used in the Create Table() directive to denote that a field will contain unique data, usually used for natural keys, foreign keys etc.

For example:

```
Create Table Employee( Emp_PKey Int Identity(1, 1) Constraint
PK_Employee_Emp_PKey Primary Key,

Emp_SSN Numeric Not Null Unique,

 Emp_FName varchar(16),

Emp_LName varchar(16) )
```

Distinct is used in the Select statement to notify the query that you only want the unique items returned when a field holds data that may not be unique.

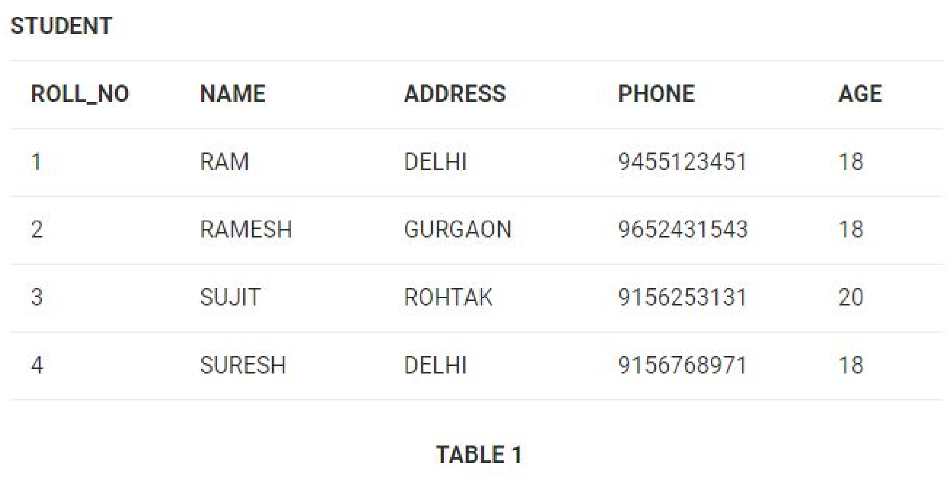
```
Select Distinct Emp_LName From Employee
```

You may have many employees with the same last name, but you only want each different last name.

Obviously if the field you are querying holds unique data, then the *Distinct* keyword becomes superfluous.

What is a Relational Database?

Relational database means the data is stored as well as retrieved in the form of relations (tables).


STUDENT

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI	9156768971	18

TABLE 1

These are some important terminologies that are used in terms of relation.
Attribute: Attributes are the properties that define a relation. e.g.; **ROLL_NO**, **NAME** etc.
Tuple: Each row in the relation is known as tuple.The above relation contains 4 tuples.
Degree: The number of attributes in the relation is known as degree of the relation. The STUDENT relation defined above has degree 5.
Cardinality: The number of tuples in a relation is known as cardinality. The **STUDENT** relation defined above has cardinality 4.
Column: Column represents the set of values for a particular attribute. The column **ROLL_NO** is extracted from relation STUDENT.

List some databases (I.Q)

SQL is the programming language for relational databases (explained below) like MySQL, Oracle, Sybase, SQL Server, Postgre, etc. Other non-relational databases (also called NoSQL) databases like MongoDB, DynamoDB, etc do not use SQL

SQL vs NoSQL (I.Q)

1. **SQL** databases are table based databases whereas **NoSQL** databases can be document based, key-value pairs, graph databases.
2. **SQL** databases are vertically scalable while **NoSQL** databases are horizontally scalable.
3. **SQL** databases have a predefined schema whereas **NoSQL** databases use dynamic schema for unstructured data.

What are Keys?

A DBMS key is an attribute or set of an attribute which helps you to identify a row(tuple) in a relation(table). They allow you to find the relation between two tables. Keys help you uniquely identify a row in a table by a combination of one or more columns in that table.

Keys in Database

Super key
Candidate key
Primary key
Alternate key
Composite primary key
Unique key
Foreign key
Compound key
Surrogate key

What is the Super key?

A superkey is a group of single or multiple keys which identifies rows in a table. A Super key may have additional attributes that are not needed for unique identification.

Example:

EmpSSN	EmpNum	Empname
9812345098	AB05	Shown
9876512345	AB06	Roslyn
199937890	AB07	James

In the above-given example, EmpSSN and EmpNum name are superkeys.

What is a Primary Key?

PRIMARY KEY is a column or group of columns in a table that uniquely identify every row in that table. The Primary Key can't be a duplicate meaning the same value can't appear more than once in the table. A table cannot have more than one primary key.

Rules for defining Primary key:

- Two rows can't have the same primary key value
- It must for every row to have a primary key value.
- The primary key field cannot be null.
- The value in a primary key column can never be modified or updated if any foreign key refers to that primary key.

Example:

In the following example, StudID is a Primary Key.

StudID	Roll No	First Name	LastName	Email
1	11	Tom	Price	abc@gmail.com
2	12	Nick	Wright	xyz@gmail.com
3	13	Dana	Natan	mno@yahoo.com

What is the Alternate key?

ALTERNATE KEYS is a column or group of columns in a table that uniquely identify every row in that table. A table can have multiple choices for a primary key but only one can be set as the primary key. All the keys which are not primary key are called an Alternate Key.

Example:

In this table, StudID, Roll No, Email are qualified to become a primary key. But since StudID is the primary key, Roll No, Email becomes the alternative key.

StudID	Roll No	First Name	LastName	Email
1	11	Tom	Price	abc@gmail.com
2	12	Nick	Wright	xyz@gmail.com
3	13	Dana	Natan	mno@yahoo.com

What is a Candidate Key?

CANDIDATE KEY is a set of attributes that uniquely identify tuples in a table. Candidate Key is a super key with no repeated attributes. The Primary key should be selected from the candidate keys. Every table must have at least a single candidate key. A table can have multiple candidate keys but only a single primary key.

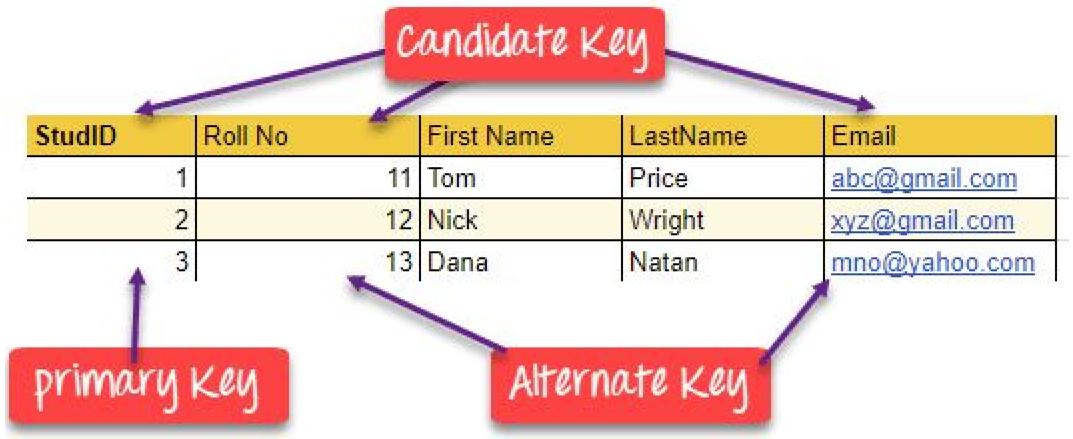
Properties of Candidate key:

- It must contain unique values
- Candidate key may have multiple attributes
- Must not contain null values
- It should contain minimum fields to ensure uniqueness

- Uniquely identify each record in a table

Example: In the given table Stud ID, Roll No, and email are candidate keys which help us to uniquely identify the student record in the table.

StudID	Roll No	First Name	LastName	Email
1	11	Tom	Price	abc@gmail.com
2	12	Nick	Wright	xyz@gmail.com
3	13	Dana	Natan	mno@yahoo.com



What is the Foreign key?

FOREIGN KEY is a column that creates a relationship between two tables. The purpose of Foreign keys is to maintain data integrity and allow navigation between two different instances of an entity. It acts as a cross-reference between two tables as it references the primary key of another table.

Example:

DeptCode	DeptName
001	Science
002	English
005	Computer

Teacher ID	Fname	Lname
B002	David	Warner
B017	Sara	Joseph
B009	Mike	Brunton

In this example, we have two table, teach and department in a school. However, there is no way to see which search work in which department.

In this table, adding the foreign key in Deptcode to the Teacher name, we can create a relationship between the two tables.

Teacher ID	DeptCode	Fname	Lname
B002	002	David	Warner
B017	002	Sara	Joseph

B009	001	Mike	Brunton

This concept is also known as Referential Integrity.

What is the Compound key?

COMPOUND KEY has two or more attributes that allow you to uniquely recognize a specific record. It is possible that each column may not be unique by itself within the database. However, when combined with the other column or columns the combination of composite keys become unique. The purpose of compound key is to uniquely identify each record in the table.

Example:

OrderNo	**PorductID**	**Product Name**	**Quantity**
B005	JAP102459	Mouse	5
B005	DKT321573	USB	10
B005	OMG446789	LCD Monitor	20
B004	DKT321573	USB	15
B002	OMG446789	Laser Printer	3

In this example, OrderNo and ProductID can't be a primary key as it does not uniquely identify a record. However, a compound key of Order ID and Product ID could be used as it uniquely identified each record.

What is the Composite key?

COMPOSITE KEY is a combination of two or more columns that uniquely identify rows in a table. The combination of columns guarantees uniqueness, though individually uniqueness is not guaranteed. Hence, they are combined to uniquely identify records in a table.

Difference between Compound key and Composite key?

The difference between compound and the composite key is that any part of the compound key can be a foreign key, but the composite key may or may not be a part of the foreign key.

What is a Surrogate Key?

An artificial key which aims to uniquely identify each record is called a surrogate key. These kinds of keys are unique because they are created when you don't have any natural primary key. They do not lend any meaning to the data in the table. Surrogate key is usually an integer.

Fname	Lastname	Start Time	End Time
Anne	Smith	09:00	18:00
Jack	Francis	08:00	17:00
Anna	McLean	11:00	20:00

Shown	Willam	14:00	23:00

Above, given example, shown shift timings of the different employee. In this example, a surrogate key is needed to uniquely identify each employee.

Surrogate keys are allowed when

- No property has the parameter of the primary key.
- In the table when the primary key is too big or complicated.

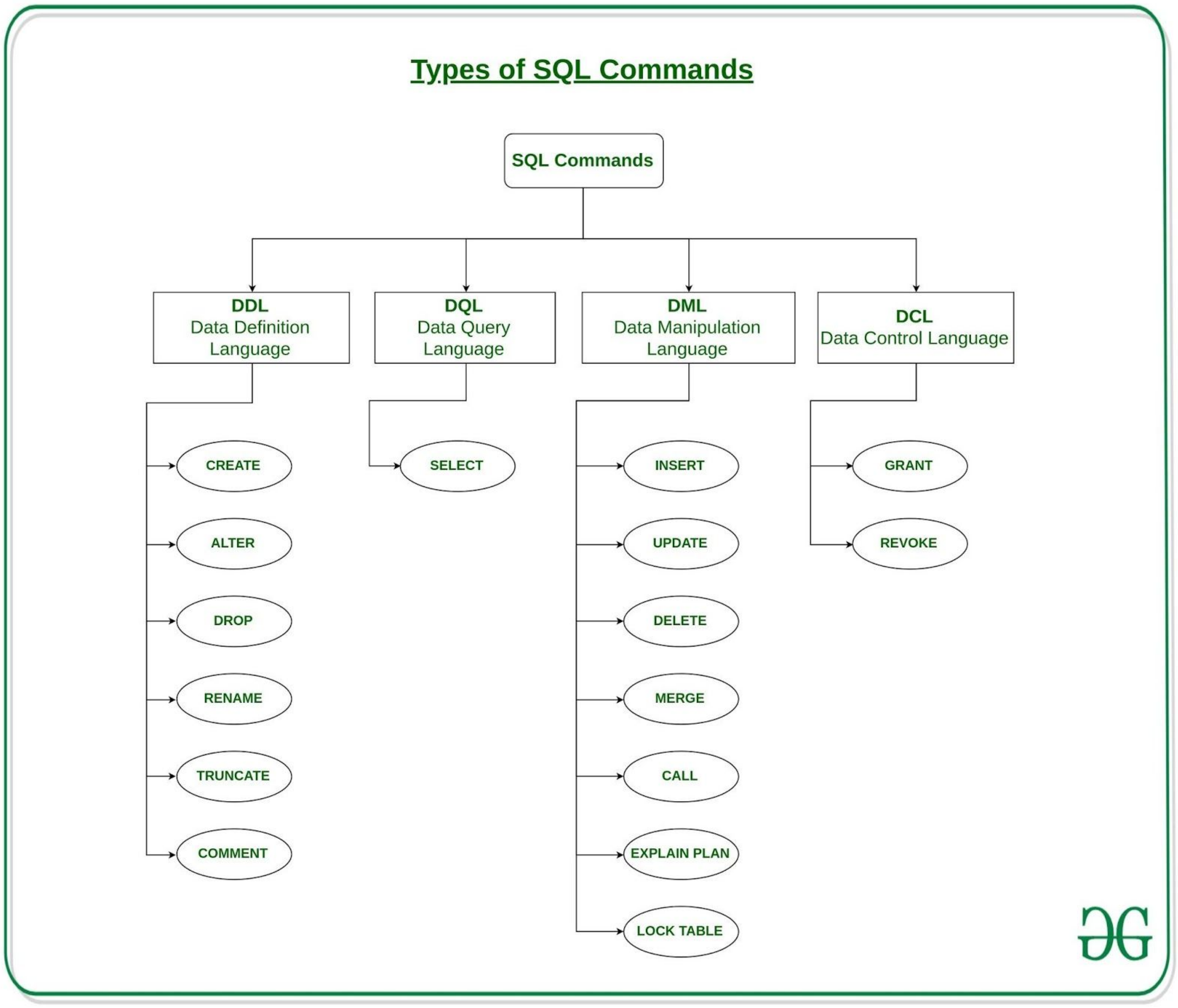
primary key vs unique key

Both Primary key and Unique Key are used to uniquely define a row in a table. Primary Key creates a clustered index of the column whereas a Unique creates an unclustered index of the column . A Primary Key doesn't allow NULL value , however a Unique Key does allow one NULL value key.

SQL commands

SQL commands are mainly categorized into five categories as:

1. DDL – Data Definition Language
2. DQL – Data Query Language
3. DML – Data Manipulation Language
4. DCL – Data Control Language
5. TCL – Transaction Control Language.


Types of SQL Commands
SQL Commands
DDL Data Definition Language
DQL Data Query Language
DML Data Manipulation Language
DCL Data Control Language
CREATE
ALTER
DROP
RENAME
TRUNCATE
COMMENT
SELECT
INSERT
UPDATE
DELETE
MERGE
CALL
EXPLAIN PLAN
LOCK TABLE
GRANT
REVOKE

1. **DDL(Data Definition Language) :** DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database.

   Examples of DDL commands:

   - **CREATE** – is used to create the database or its objects (like table, index, function, views, store procedure and triggers).
   - **DROP** – is used to delete objects from the database.
   - **ALTER**-is used to alter the structure of the database.
   - **TRUNCATE**–is used to remove all records from a table, including all spaces allocated for the records are removed.
   - **COMMENT** –is used to add comments to the data dictionary.
   - **RENAME** –is used to rename an object existing in the database.

2. **DQL (Data Query Language) :**

   DML statements are used for performing queries on the data within schema objects. The purpose of DQL Command is to get some schema relation based on the query passed to it.

   Example of DQL:

   - **SELECT** – is used to retrieve data from the a database.

3. **DML(Data Manipulation Language) :** The SQL commands that deals with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements.

   Examples of DML:

   - **INSERT** – is used to insert data into a table.

- **UPDATE** – is used to update existing data within a table.
- **DELETE** – is used to delete records from a database table.

Truncate vs Delete vs DROP?

- The DROP command removes a table from the database. All the tables' rows, indexes, and privileges will also be removed. The operation cannot be rolled back.
- DROP and TRUNCATE are DDL commands, whereas DELETE is a DML command.
- DELETE operations can be rolled back (undone), while DROP and TRUNCATE operations cannot be rolled back.
- Truncate reinitializes the identity by making changes in data definition therefore it is DDL, whereas Delete only deletes the records from the table and doesn't make any changes in its Definition that's why it is DML.

- TRUNCATE TABLE statement drop and re-create the table in such a way that any auto-increment value is reset to its start value which is generally 1.

- DELETE lets you filter which rows to be deleted based upon an optional WHERE clause, whereas TRUNCATE TABLE doesn't support WHERE clause it just removes all the rows.

- TRUNCATE TABLE is faster and uses fewer system resources than DELETE, because DELETE scans the table to generate a count of rows that were affected then delete the rows one by one and records an entry in the database log for each deleted row, while TRUNCATE TABLE just delete all the rows without providing any additional information.

4. **DCL(Data Control Language) :** DCL includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system.

   Examples of DCL commands:

   - **GRANT** gives user's access privileges to the database.

  - **REVOKE**-withdraw user's access privileges given by using the GRANT command.

5. **TCL(transaction Control Language) :** TCL commands deals with the transaction within the database.

   Examples of TCL commands:

   - **COMMIT**– commits a Transaction.
   - **ROLLBACK**– rollbacks a transaction in case of any error occurs.
   - **SAVEPOINT**–sets a savepoint within a transaction.
   - **SET TRANSACTION**–specify characteristics for the transaction

Transaction

Transactions group a set of tasks into a single execution unit. Each transaction begins with a specific task and ends when all the tasks in the group successfully complete. If any of the tasks fail, the transaction fails. Therefore, a transaction has only two results: success or failure. Incomplete steps result in the failure of the transaction. A database transaction, by definition, must be atomic, consistent, isolated and durable. These are popularly known as ACID properties.

What are ACID properties?

Atomicity − This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed. States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.

Consistency − The database must remain in a consistent state after any transaction. No transaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.

Isolation − In a database system where more than one transaction is being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.

Durability − The database should be durable enough to hold all its latest updates even if the system fails or restarts. If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.

How would you implement a rollback function if you are creating a DBMS?

```
SELECT @BookCount = COUNT(*) FROM Books WHERE name = 'Book15'

 IF @BookCount > 1

  BEGIN
    ROLLBACK TRANSACTION AddBook
    PRINT 'A book with the same name already exists'
  END

 ELSE
  BEGIN
    COMMIT TRANSACTION AddStudent
    PRINT 'New book added successfully'
  END
```

SQL Views

Views in SQL are a kind of **virtual table**. A view also has rows and columns as they are in a real table in the database. We can create a view by selecting fields from one or more tables present in the database. A View can either have all the rows of a table or specific rows based on certain conditions.
Views account for logical data independence as the growth and restructuring of base tables are not reflected in views.

Advantages of Views:

- As there is no physical location where the data in the view is stored, it generates output without wasting resources.

- Data access is restricted as it does not allow commands like insertion, updation, and deletion.

Disadvantages of Views:

- The view becomes irrelevant if we drop a table related to that view.
- Much memory space is occupied when the view is created for large tables.

Creating View from a single table:

In this example we will create a View named DetailsView from the table StudentDetails.

Query:

```
CREATE VIEW DetailsView AS
SELECT NAME, ADDRESS
FROM StudentDetails
WHERE S_ID < 5;
```

To see the data in the View, we can query the view in the same manner as we query a table.

SELECT * FROM DetailsView;

What are Triggers?

Trigger: A trigger is a stored procedure in a 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 trigger [trigger_name]

[before | after]

{insert | update | delete}

on [table_name]

[for each row]

[trigger_body]
```

Explanation of syntax:

1. create trigger [trigger_name]: Creates or replaces an existing trigger with the trigger_name.

2. [before | after]: This specifies when the trigger will be executed.
3. {insert | update | delete}: This specifies the DML operation.
4. on [table_name]: This specifies the name of the table associated with the trigger.
5. [for each row]: This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected.
6. [trigger_body]: This provides the operation to be performed as trigger is fired

BEFORE and AFTER of Trigger:

BEFORE triggers run the trigger action before the triggering statement is run.

AFTER triggers run the trigger action after the triggering statement is run.

Difference between function and stored procedure

https://www.geeksforgeeks.org/sql-difference-between-functions-and-stored-procedures-in-pl-sql/

Indexes

Indexes are special lookup tables that the database search engine can use to speed up data retrieval. Simply put, an index is a pointer to data in a table. An index in a database is very similar to an index in the back of a book.

For example, if you want to reference all pages in a book that discusses a certain topic, you first refer to the index, which lists all the topics alphabetically and are then referred to one or more specific page numbers.

An index helps to speed up SELECT queries and WHERE clauses, but it slows down data input, with the UPDATE and the INSERT statements. Indexes can be created or dropped with no effect on the data.

Creating an index involves the CREATE INDEX statement, which allows you to name the index, to specify the table and which column or columns to index, and to indicate whether the index is in an ascending or descending order.

Indexes can also be unique, like the UNIQUE constraint, in that the index prevents duplicate entries in the column or combination of columns on which there is an index.

The CREATE INDEX Command
The basic syntax of a CREATE INDEX is as follows.

```
CREATE INDEX index_name ON table_name;
```

Single-Column Indexes

A single-column index is created based on only one table column. The basic syntax is as follows.

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

Unique Indexes

Unique indexes are used not only for performance, but also for data integrity. A unique index does not allow any duplicate values to be inserted into the table. The basic syntax is as follows.

```
CREATE UNIQUE INDEX index_name
on table_name (column_name);
```

Composite Indexes

A composite index is an index on two or more columns of a table. Its basic syntax is as follows.

```
CREATE INDEX index_name
on table_name (column1, column2);
```

Whether to create a single-column index or a composite index, take into consideration the column(s) that you may use very frequently in a query's WHERE clause as filter conditions.

Should there be only one column used, a single-column index should be the choice. Should there be two or more columns that are frequently used in the WHERE clause as filters, the composite index would be the best choice.

Implicit Indexes

Implicit indexes are indexes that are automatically created by the database server when an object is created. Indexes are automatically created for primary key constraints and unique constraints

.

The DROP INDEX Command

An index can be dropped using SQL DROP command. Care should be taken when dropping an index because the performance may either slow down or improve.

The basic syntax is as follows −

```
DROP INDEX index_name;
```

When should indexes be avoided?

Although indexes are intended to enhance a database's performance, there are times when they should be avoided.

The following guidelines indicate when the use of an index should be reconsidered.

- Indexes should not be used on small tables.
- Tables that have frequent, large batch updates or insert operations.
- Indexes should not be used on columns that contain a high number of NULL values.
- Columns that are frequently manipulated should not be indexed.

Advantages of indexes

1. Speed up SELECT query
2. Helps to make a row unique or without duplicates(primary,unique)

3. If the index is set to full-text index, then we can search against large string values. for example to find a word from a sentence etc.

Disadvantages of indexes

1. Indexes take additional disk space.
2. indexes slow down INSERT,UPDATE and DELETE, but will speed up UPDATE if the WHERE condition has an indexed field. INSERT, UPDATE and DELETE become slower because on each operation the indexes must also be updated.

clustered indexes versus unclustered indexes

A clustered index defines the order in which data is physically stored in a table. Table data can be sorted in only one way, therefore, there can be only one clustered index per table. In SQL Server, the primary key constraint automatically creates a clustered index on that particular column.

A non-clustered index doesn't sort the physical data inside the table. In fact, a non-clustered index is stored at one place and table data is stored in another place. This is similar to a textbook where the book content is located in one place and the index is located in another. This allows for more than one non-clustered index per table.

https://www.sqlshack.com/what-is-the-difference-between-clustered-and-non-clustered-indexes-in-sql-server/