## NVL vs NVL2

These functions work with any data type and pertain to the use of null values in the expression list.

These are all single-row functions i.e. provide one result per row.

NVL(expr1, expr2): In SQL, NVL() converts a null value to an actual value. Data types that can be used are date, character, and number. Data types must match with each other. i.e. expr1 and expr2 must be of the same data type.

Syntax:

NVL (expr1, expr2)

NVL2(expr1, expr2, expr3): The NVL2 function examines the first expression. If the first expression is not null, then the NVL2 function returns the second expression. If the first expression is null, then the third expression is returned i.e. If expr1 is not null, NVL2 returns expr2. If expr1 is null, NVL2 returns expr3. The argument expr1 can have any data type.

Syntax: NVL2 (expr1, expr2, expr3)

## Narrow and Wide format

## Normalization

Ever wondered how databases organize their data efficiently to prevent redundancy and ensure reliability? Let's delve into the world of normalization, where we explore the art of structuring data to optimize database performance and maintain data integrity!

Normalization is the process of organizing data in a database to reduce redundancy and dependency by dividing large tables into smaller, related tables.

Normalization helps organize tables and their relationships to achieve optimal database design.

Why is Normalization Required?

Normalization is required to:

Minimize Redundancy: Reduces data duplication, leading to storage efficiency.

Eliminate Update Anomalies: Ensures data consistency when updating records. Modifications are made in only one place, avoiding inconsistent data.

Prevent Insertion Anomalies: Facilitates adding new records without unnecessary duplication. Ensures that adding new data does not require adding unrelated data.

Avoid Deletion Anomalies: Allows for removal of data without unintended loss of information. Removal of data does not cause unintended loss of information in related tables.

Consider a denormalized table of `Employees`:

A screenshot of a computer

Description automatically generated

### Explanation of Normalization Benefits

1. \*\*Minimize Redundancy\*\*

- \*\*Issue:\*\* Redundant data leads to inefficient storage.

- \*\*Example:\*\* In the table above, both `Manager\_ID` and `Manager\_Name` are repeated for each employee, causing redundancy.

- \*\*Solution:\*\* Normalize by creating separate tables for `Employees`, `Departments`, and `Managers` to store data without repetition.

2. \*\*Eliminate Update Anomalies\*\*

- \*\*Issue:\*\* Modifying data in one place but not in related places can lead to inconsistencies.

- \*\*Example:\*\* If John's name (`Manager\_Name`) changes to "Johnny", every occurrence of "John" in `Manager\_Name` needs updating.

- \*\*Solution:\*\* Normalize by having `Manager\_ID` as a foreign key in the `Employees` table, referencing a `Managers` table where names are stored uniquely.

3. \*\*Prevent Insertion Anomalies\*\*

- \*\*Issue:\*\* Adding new data requires inserting the same information multiple times.

- \*\*Example:\*\* To add a new department with a new manager, one must repeat the manager's information across multiple employee entries.

- \*\*Solution:\*\* Normalize by having separate tables for `Employees` and `Managers`, linked by foreign keys, so adding a new employee or manager does not duplicate unrelated data.

4. \*\*Avoid Deletion Anomalies\*\*

- \*\*Issue:\*\* Removing data unintentionally causes loss of related information.

- \*\*Example:\*\* Deleting an employee might inadvertently delete their department or manager information.

- \*\*Solution:\*\* Normalize by establishing relationships between tables (`Employees` to `Managers` via `Manager\_ID`) so that deleting an employee does not affect unrelated data like department or manager information.

### Normalized Structure

\*\*Employees:\*\*

A screenshot of a computer

Description automatically generated

\*\*Managers:\*\*

A screenshot of a computer

Description automatically generated

\*\*Departments:\*\*

A screenshot of a computer

Description automatically generated

Below are the Different Normal Forms (NF) in SQL:

1. \*\*First Normal Form (1NF):\*\* Ensures each column contains atomic (indivisible) values and there are no repeating groups or arrays.

2. Second Normal Form (2NF):\*\* Meets 1NF requirements and ensures all non-key attributes are fully dependent on the primary key, eliminating partial dependencies.

3. \*\*Third Normal Form (3NF):\*\* Meets 2NF requirements and ensures that no transitive dependencies exist, meaning no non-key column is dependent on another non-key column.

4. \*\*Boyce-Codd Normal Form (BCNF):\*\* A stricter version of 3NF where every determinant must be a candidate key, ensuring all functional dependencies are satisfied.

5. \*\*Fourth Normal Form (4NF):\*\* Addresses multi-valued dependencies where a table has more than one multi-valued dependency, ensuring no independent multi-valued facts within a single table.

6. \*\*Fifth Normal Form (5NF):\*\* Ensures that a database is free from join dependencies and anomalies stemming from join operations.

## First Normal Form (1NF)

Ever wondered how databases ensure data is neatly organized without any messy duplicates or awkward groupings? Let's dive into the First Normal Form (1NF), where we explore the foundational rule that makes sure every piece of data plays nicely in its own space!

First Normal Form (1NF) ensures that each column in a table contains atomic (indivisible) values and there are no repeating groups of columns.

### Explanation of First Normal Form (1NF):

1. \*\*Atomic Values\*\*: Every column in a table must contain atomic (indivisible) values. This means that a column should not have multiple values or repeating groups.

2. \*\*Unique Column Names\*\*: Each column in a table must have a unique name. This ensures clarity and avoids ambiguity in data retrieval and updates.

3. \*\*No Repeating Groups\*\*: Each column should contain data that directly relates to the primary key, and each row should be unique.

### Example Input Table:

Consider a table that stores information about students and the courses they are enrolled in. A non-1NF table might look like this:

A white rectangular object with a black border

Description automatically generated with medium confidence

This table violates 1NF because:

- The `Courses` column contains multiple values (comma-separated list), violating the atomicity rule.

- It's not immediately clear which course belongs to which student without parsing the `Courses` column.

To convert the given example input table into the First Normal Form (1NF), we need to ensure that each column contains atomic values and that there are no repeating groups. Here's how the table would look in 1NF:

\*\*Student Table:\*\*

A screenshot of a computer

Description automatically generated

In this 1NF table:

- Each column (`Student ID`, `Student Name`, `Course`) contains atomic values.

- There are no repeating groups, and each row represents a unique combination of student and course.

This structure adheres to the principles of 1NF by organizing data into a format that supports efficient querying, data integrity, and easier maintenance.

### Importance of First Normal Form (1NF):

1. \*\*Data Integrity\*\*: Ensures each piece of data is stored in a structured, clear format, reducing the risk of inconsistencies and errors.

2. \*\*Efficient Querying\*\*: Allows for efficient querying and indexing of data, as each column contains only single, atomic values.

3. \*\*Flexibility\*\*: Facilitates easier updates and modifications to the database schema without the need for complex data restructuring.

4. \*\*Standardization\*\*: Provides a standardized way of organizing data, making it easier for developers and users to understand and work with the database.

By adhering to 1NF, databases become more robust and easier to maintain, ensuring that data remains accurate and accessible over time.

## Second Normal Form (2NF)

How can a slight tweak in database design turn confusion into clarity? Let's dive into the Second Normal Form (2NF) and uncover its transformative power!

Second Normal Form (2NF) ensures that a table is in First Normal Form (1NF) and that all non-key attributes are fully functionally dependent on the primary key.

### Explanation of Second Normal Form (2NF):

Second Normal Form (2NF) builds upon First Normal Form (1NF) by addressing issues related to partial dependencies within a table.

1. \*\*First Normal Form (1NF) Recap\*\*: Ensures atomicity of columns (no multi-valued attributes) and unique column names.

2. \*\*Second Normal Form (2NF) Requirement\*\*: In addition to 1NF, every non-key column must be fully functionally dependent on the entire primary key.

Example Input Table and Output:

Consider a table that stores information about students and the courses they are enrolled in, including information about the instructors:

\*\*Input Table (Not in 2NF):\*\*

A screenshot of a computer

Description automatically generated

This table is not in 2NF because:

- The `Instructor` column depends only on the `Course`, not the entire primary key (`Student ID`, `Course`).

\*\*Output Table (In 2NF):\*\*

To normalize this table into 2NF, we would split it into two tables:

\*\*Students Table:\*\*

A screenshot of a computer

Description automatically generated

\*\*Courses Table:\*\*

A screenshot of a computer

Description automatically generated

### Importance of Second Normal Form (2NF):

1. \*\*Eliminates Redundancy\*\*: By organizing data into separate tables, each containing a unique set of information, redundancy is minimized, and data consistency is improved.

2. \*\*Maintains Data Integrity\*\*: Ensures that each piece of data is stored in only one place, reducing the risk of anomalies such as update anomalies (where updating one piece of information leads to inconsistencies).

3. \*\*Improves Query Performance\*\*: With well-defined relationships between tables, querying becomes more efficient, as data retrieval does not require scanning unnecessary columns.

4. \*\*Facilitates Scalability\*\*: Allows for easier expansion and modification of the database schema as new requirements arise, without the need for extensive restructuring.

By understanding and implementing Second Normal Form (2NF), database designs become more robust, efficient, and adaptable to changing needs, making data management more effective in both small and large-scale applications.

## Third Normal Form (3NF)

Ever wondered how database design can empower data integrity and streamline information retrieval? Let's unravel the transformative principles of Third Normal Form (3NF) and discover its impact on shaping efficient databases!

Third Normal Form (3NF) ensures that a table is in Second Normal Form (2NF) and eliminates transitive dependencies by requiring that non-key attributes depend only on the primary key.

### Explanation of Third Normal Form (3NF):

Third Normal Form (3NF) further refines database design by addressing transitive dependencies within tables. Here's a beginner-friendly explanation along with an example:

1. \*\*First Normal Form (1NF) and Second Normal Form (2NF) Recap\*\*: Ensures atomicity of columns and eliminates partial dependencies respectively.

2. \*\*Third Normal Form (3NF) Requirement\*\*: In addition to 2NF, every non-key column must depend only on the primary key (no transitive dependencies).

### Example Input Table and Output:

Consider a table that stores information about employees, their departments, and the locations of those departments:

\*\*Input Table (Not in 3NF):\*\*

A screenshot of a computer

Description automatically generated

This table is not in 3NF because:

- There's a transitive dependency where `Department Location` depends on `Department`, which is not part of the primary key (`Employee ID`).

\*\*Output Table (In 3NF):\*\*

To normalize this table into 3NF, we would split it into two tables:

\*\*Employees Table:\*\*

A screenshot of a computer

Description automatically generated

\*\*Departments Table:\*\*

A screenshot of a phone

Description automatically generated

\*\*EmployeeDepartments Table:\*\*

A screenshot of a computer

Description automatically generated

### Importance of Third Normal Form (3NF):

1. \*\*Reduces Redundancy\*\*: By organizing data into separate tables and ensuring each table has a distinct purpose (no transitive dependencies), redundancy is minimized, leading to more efficient storage and maintenance of data.

2. \*\*Enhances Data Integrity\*\*: Prevents anomalies such as insertion, update, and deletion anomalies that can occur when data dependencies are not properly managed.

3. \*\*Simplifies Database Maintenance\*\*: With well-defined relationships and clear dependencies, database administrators can more easily modify and extend the database schema as business requirements evolve.

4. \*\*Improves Query Efficiency\*\*: Queries become more straightforward and efficient as they only need to access relevant tables and columns, reducing the complexity of joins and improving overall performance.

By adhering to Third Normal Form (3NF), databases are structured in a way that supports data integrity, flexibility, and efficiency, making them more robust and suitable for scalable applications.

## Boyce-Codd Normal Form (BCNF)

Did you know that there's a higher level of database normalization that goes beyond Third Normal Form (3NF) to ensure even stricter data integrity? Let's uncover the transformative power of Boyce-Codd Normal Form (BCNF) and how it revolutionizes database design!

Boyce-Codd Normal Form (BCNF) ensures that a table is in Third Normal Form (3NF) and that every determinant in the table is a candidate key.

Explanation of Boyce-Codd Normal Form (BCNF):

Boyce-Codd Normal Form (BCNF) is a stricter form of normalization that ensures every determinant (attributes determining other attributes) in a table is a candidate key. Here's a simplified explanation along with an example:

1. \*\*First Normal Form (1NF), Second Normal Form (2NF), and Third Normal Form (3NF) Recap\*\*: Address atomicity of columns, eliminate partial dependencies, and eliminate transitive dependencies respectively.

2. \*\*Boyce-Codd Normal Form (BCNF) Requirement\*\*: In addition to 3NF, every determinant must be a candidate key, meaning there are no non-trivial functional dependencies of attributes on anything other than a candidate key.

### Example Input Table and Output:

Consider a table that stores information about employees and the projects they are assigned to, including the project managers:

\*\*Input Table (Not in BCNF):\*\*

A screenshot of a project

Description automatically generated

This table is not in BCNF because:

- There's a dependency where `Project Manager` depends on `Project ID`, which is not a candidate key by itself.

\*\*Output Table (In BCNF):\*\*

To normalize this table into BCNF, we would typically break it down into multiple tables:

\*\*Employees Table:\*\*

A screenshot of a computer

Description automatically generated

\*\*Projects Table:\*\*

A screenshot of a project

Description automatically generated

\*\*ProjectAssignments Table:\*\*

A screenshot of a computer

Description automatically generated

\*\*ProjectManagers Table:\*\*

A screenshot of a computer

Description automatically generated

### Importance of Boyce-Codd Normal Form (BCNF):

1. \*\*Eliminates Redundancy and Anomalies\*\*: By ensuring that every determinant is a candidate key, BCNF eliminates the possibility of update, insertion, and deletion anomalies that can occur in less normalized forms.

2. \*\*Maintains Data Integrity\*\*: With clearly defined dependencies and relations between tables, BCNF ensures that data remains accurate and consistent throughout the database.

3. \*\*Improves Query Efficiency\*\*: Queries can be optimized since tables are structured in a way that minimizes unnecessary joins and ensures that only relevant data is accessed.

4. \*\*Supports Scalability and Flexibility\*\*: BCNF facilitates easier database maintenance and schema modifications as the application evolves, ensuring the database remains efficient and adaptable to changing business needs.

By adhering to Boyce-Codd Normal Form (BCNF), databases are not only more robust and reliable but also more efficient in managing and retrieving data, making them suitable for complex and large-scale applications.

## Fourth Normal Form (4NF)

- Addresses multi-valued dependencies.

- Example: Splitting multi-valued attributes into separate tables.

## Fifth Normal Form (5NF)

- Deals with cases where a non-trivial join dependency exists.

- Example: Further decomposing tables to remove join dependencies.

## concatenation operator

In SQL for appending two strings, the "concatenation operator" is used and its symbol is " || ".

## Database Architecture

Database architecture refers to the design and structure of a database system, including the arrangement and interaction of its components. There are several different types of database architectures, each suited to different use cases and performance requirements. Here are some common types:

### 1. \*\*Single-Tier Architecture\*\*

\*\*Description\*\*:

- In a single-tier architecture, the database system consists of a single layer where the database and the application reside on the same system.

\*\*Use Cases\*\*:

- Suitable for small, simple applications with low data volume.

- Often used in standalone applications or small-scale desktop applications.

\*\*Advantages\*\*:

- Simple to design and implement.

- Low latency due to proximity of application and database.

\*\*Disadvantages\*\*:

- Limited scalability.

- Poor performance for larger datasets or higher user loads.

### 2. \*\*Two-Tier Architecture\*\*

\*\*Description\*\*:

- The two-tier architecture separates the database system into two layers: the client layer and the server layer.

- The client layer contains the application and the user interface, while the server layer contains the database management system (DBMS).

\*\*Use Cases\*\*:

- Suitable for small to medium-sized applications.

- Commonly used in client-server applications.

\*\*Advantages\*\*:

- Improved performance compared to single-tier architecture.

- Better separation of concerns between the client and server.

\*\*Disadvantages\*\*:

- Scalability is still limited.

- Increased complexity in managing client-server communication.

### 3. \*\*Three-Tier Architecture\*\*

\*\*Description\*\*:

- The three-tier architecture introduces an additional middle layer between the client and server, often called the application server or business logic layer.

- The three layers are: the presentation layer (client), the application layer (business logic), and the data layer (database).

\*\*Use Cases\*\*:

- Suitable for medium to large-scale applications.

- Common in web applications and enterprise systems.

\*\*Advantages\*\*:

- Better scalability and performance due to the separation of concerns.

- Improved security as the business logic layer can act as a firewall between the client and the database.

- Easier to maintain and update each layer independently.

\*\*Disadvantages\*\*:

- More complex to design and implement.

- Potentially higher latency due to additional layer of communication.

### 4. \*\*N-Tier Architecture\*\*

\*\*Description\*\*:

- N-tier architecture extends the three-tier architecture by adding more layers, such as caching servers, load balancers, and additional application servers.

- The number of tiers can vary based on the complexity and requirements of the application.

\*\*Use Cases\*\*:

- Suitable for large-scale, highly distributed applications.

- Commonly used in complex enterprise systems and cloud-based applications.

\*\*Advantages\*\*:

- Highly scalable and flexible.

- Improved fault tolerance and load balancing.

- Enhanced security and separation of concerns.

\*\*Disadvantages\*\*:

- Very complex to design, implement, and manage.

- Increased latency due to multiple layers of communication.

### 5. \*\*Distributed Database Architecture\*\*

\*\*Description\*\*:

- In distributed database architecture, the database is spread across multiple physical locations, often on different servers or even in different geographical areas.

- Can be either homogenous (same DBMS at all sites) or heterogeneous (different DBMS at different sites).

\*\*Use Cases\*\*:

- Suitable for applications requiring high availability and fault tolerance.

- Commonly used in global applications and large-scale systems.

\*\*Advantages\*\*:

- Improved reliability and availability.

- Enhanced performance through data localization and load distribution.

- Better fault tolerance and disaster recovery.

\*\*Disadvantages\*\*:

- Increased complexity in design and management.

- Challenges in maintaining data consistency and integrity.

### 6. \*\*Cloud Database Architecture\*\*

\*\*Description\*\*:

- Cloud database architecture involves databases hosted on cloud platforms, such as Amazon Web Services (AWS), Google Cloud Platform (GCP), or Microsoft Azure.

- Can be based on various underlying architectures (e.g., two-tier, three-tier, distributed).

\*\*Use Cases\*\*:

- Suitable for applications requiring scalability, flexibility, and reduced infrastructure management.

- Common in modern web applications, mobile apps, and data analytics.

\*\*Advantages\*\*:

- High scalability and flexibility.

- Reduced infrastructure costs and management overhead.

- Built-in redundancy, backup, and disaster recovery features.

\*\*Disadvantages\*\*:

- Dependency on cloud service providers.

- Potential security and compliance concerns.

- Latency and performance variability depending on network conditions.

### Conclusion

Each database architecture has its strengths and weaknesses, making them suitable for different scenarios. The choice of architecture depends on factors such as application requirements, expected load, data volume, and performance needs. Understanding these architectures helps in designing efficient and scalable database systems tailored to specific use cases.

## Char vs varchar vs varchar 2

Both of these data types are used for characters, but varchar2 is used for character strings of variable length, whereas char is used for character strings of fixed length. For example, if we specify the type as char(5) then we will not be allowed to store a string of any other length in this variable, but if we specify the type of this variable as varchar2(5) then we will be allowed to store strings of variable length. We can store a string of length 3 or 4 or 2 in this variable.

## Left & Right



## NULL values vs zero vs blank space

In SQL, zero or blank space can be compared with another zero or blank space. whereas one null may not be equal to another null. null means data might not be provided or there is no data.

## ###### Interview Important ######

## EXCEPT

This operation includes tuples that are present in one relationship but should not be present in another relationship. For example: To find customers who have an account but no loan at the bank:

SELECT CustomerName FROM Depositor

EXCEPT

SELECT CustomerName FROM Borrower;

The Except operation automatically eliminates the duplicates. If all the duplicates are supposed to be retained, EXCEPT ALL is used in place of EXCEPT.

## Relationships in SQL

There are various relationships, namely:

• One-to-One Relationship.

• One to Many Relationships.

• Many to One Relationship.

• Self-Referencing Relationship

## Database Relationships

One to One

One to Many

Many to Many

Icons/Logos

### ER Diagram

Logos

## CURRENT\_DATE

## INTERVAL

## Local and global variables

Global Variable: In contrast, global variables are variables that are defined outside of functions. These variables have global scope, so they can be used by any function without passing them to the function as parameters.

Local Variable: Local variables are variables that are defined within functions. They have local scope, which means that they can only be used within the functions that define them

## Important Topics

8. \*Functions:\*

- Scalar functions (e.g., CONCAT, DATE functions)

- CASE expressions

- Window functions (e.g., ROW\_NUMBER, RANK)

13. \*Advanced SQL Concepts:\*

- Recursive queries

- Pivot and Unpivot operations

- Working with JSON data

14. \*Database Security:\*

- User management and permissions

- SQL injection prevention

15. \*Working with Multiple Databases:\*

- Connecting to and querying multiple databases

- Cross-database queries

16. \*NoSQL Databases (Optional):\*

- Basics of NoSQL databases (e.g., MongoDB, Cassandra)

17. \*SQL Best Practices:\*

- Writing efficient SQL queries

- Data modeling best practices