**DATABASE**

**SQL VS No SQL**

|  |  |  |
| --- | --- | --- |
|  | **SQL** | **NoSQL** |
| **Overview** | Relational Data | Non-relational structure |
|  |  | Utilizes a dynamic schema |
| **Scaling** | Vertical or Sharding | NoSQL databases scale better horizontally |
|  |  | Add additonal servers for load |
| **Properties** | ACID | **Consistency:** Every request receives either the most recent result or an error |
|  |  | **Availability:** Every request has a non-error result. |
|  |  | **Partition tolerance:** Any delays or losses between nodes do not interrupt the system operation. |
| **Structure** | Rows and Table | **Column-oriented:** where data is stored in cells grouped in a virtually unlimited number of columns rather than rows. |
|  |  | **Key-value stores:** which use an associative array (also known as a dictionary or map) as their data model |
|  |  | **Document stores:** which use documents to hold and encode data in standard formats, including XML, YAML, JSON (JavaScript Object Notation) and BSON. A benefit is that documents within a single database can have different data types. |
|  |  | **Graph databases**: which represent data on a graph that shows how different sets of data relate to each other |

**Normal Forms**

|  |  |  |
| --- | --- | --- |
| **1NF** | Each Column Unique | Each Cell a Single Value |
| **2NF** | Each Column directly related to primary key |  |
| **3NF** | All non-key attributes independent of each other | Each column directly related to primary key and not other columns |

**ACID**

|  |  |
| --- | --- |
| **Atomicity** | Each statement in a transaction CRUD is treated as a single unit. Either the entire statement is executed, or none of it is executed. |
| **Consistency** | ensures that transactions only make changes to tables in predefined, predictable ways |
| **Isolation** | when multiple users are reading and writing from the same table all at once, isolation of their transactions ensures that the concurrent transactions don't interfere with or affect one another. Each request can occur as though they were occurring one by one, even though they're actually occurring simultaneously |
| **Durability** | ensures that changes to your data made by successfully executed transactions will be saved, even in the event of system failure. |

**SQL**

|  |  |  |
| --- | --- | --- |
| **Keyword** | **Summary** | **Syntax** |
| Distinct | select single value from duplicate values | DISTINCT |
| Order By | Sort in ascending or descending order | SELECT \* FROM Products ORDER BY Price; |
| And | Filter on multiple conditions | WHERE Country = 'Spain' AND CustomerName LIKE 'G%'; |
| Or | Filter on multiple conditions | WHERE Country = 'Germany' OR Country = 'Spain'; |
| Not | Filter remove | WHERE NOT Country = 'Spain'; |
| Null | Find null values | SELECT CustomerName, ContactName, Address FROM Customers WHERE Address IS NULL; |
| Not Null | Has a value | SELECT column\_names FROM table\_name WHERE column\_name IS NOT NULL; |
| Min |  | SELECT MIN(Price) FROM Products; |
|  |  | SELECT MIN(Price) AS SmallestPrice FROM Products; |
| Max |  |  |
| Count |  | SELECT COUNT(ProductName) FROM Products; |
| Sum |  | SELECT SUM(Quantity) FROM OrderDetails; |
| Average |  | SELECT AVG(Price) FROM Products; |
| Like |  | SELECT \* FROM Customers WHERE CustomerName LIKE 'a%' |
| Wildcard (\_) | \_ wildcard represents a single character | SELECT \* FROM Customers WHERE city LIKE 'L\_nd\_\_'; |
| Wildcard (%) | % wildcard represents any number of characters, even zero characters | **Where it contains the letter L** SELECT \* FROM Customers WHERE city LIKE '%L%'; |
|  |  | **Starts with L** SELECT \* FROM Customers WHERE CustomerName LIKE 'La%'; |
| In | The IN operator is a shorthand for multiple OR conditions | SELECT \* FROM Customers WHERE Country IN ('Germany', 'France', 'UK'); |
| Between | Selects values within a given range | SELECT \* FROM Products WHERE Price BETWEEN 10 AND 20; |
| Alias | SQL aliases are used to give a table, or a column in a table, a temporary name. | SELECT CustomerID AS ID FROM Customers; |
| Inner Join |  | SELECT ProductID, ProductName, CategoryName FROM Products INNER JOIN Categories ON Products.CategoryID = Categories.CategoryID; |
| Stored Procedure |  | CREATE PROCEDURE SelectAllCustomers AS SELECT \* FROM Customers GO; |

|  |
| --- |
| **Stored Procedure** |
| CREATE PROCEDURE procedure\_name AS sql\_statement GO; |
| CREATE PROCEDURE SelectAllCustomers AS SELECT \* FROM Customers GO; |

**PL-SQL**

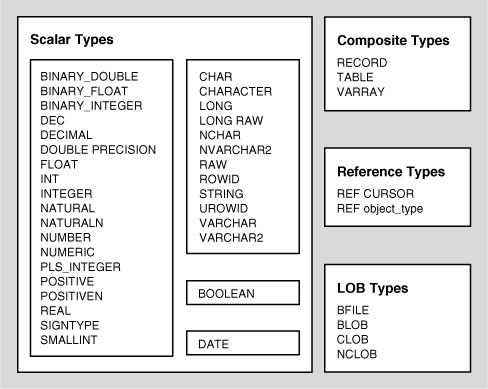
**Data Types**

A **scalar** type has no internal components. It holds a single value, such as a number or character string.

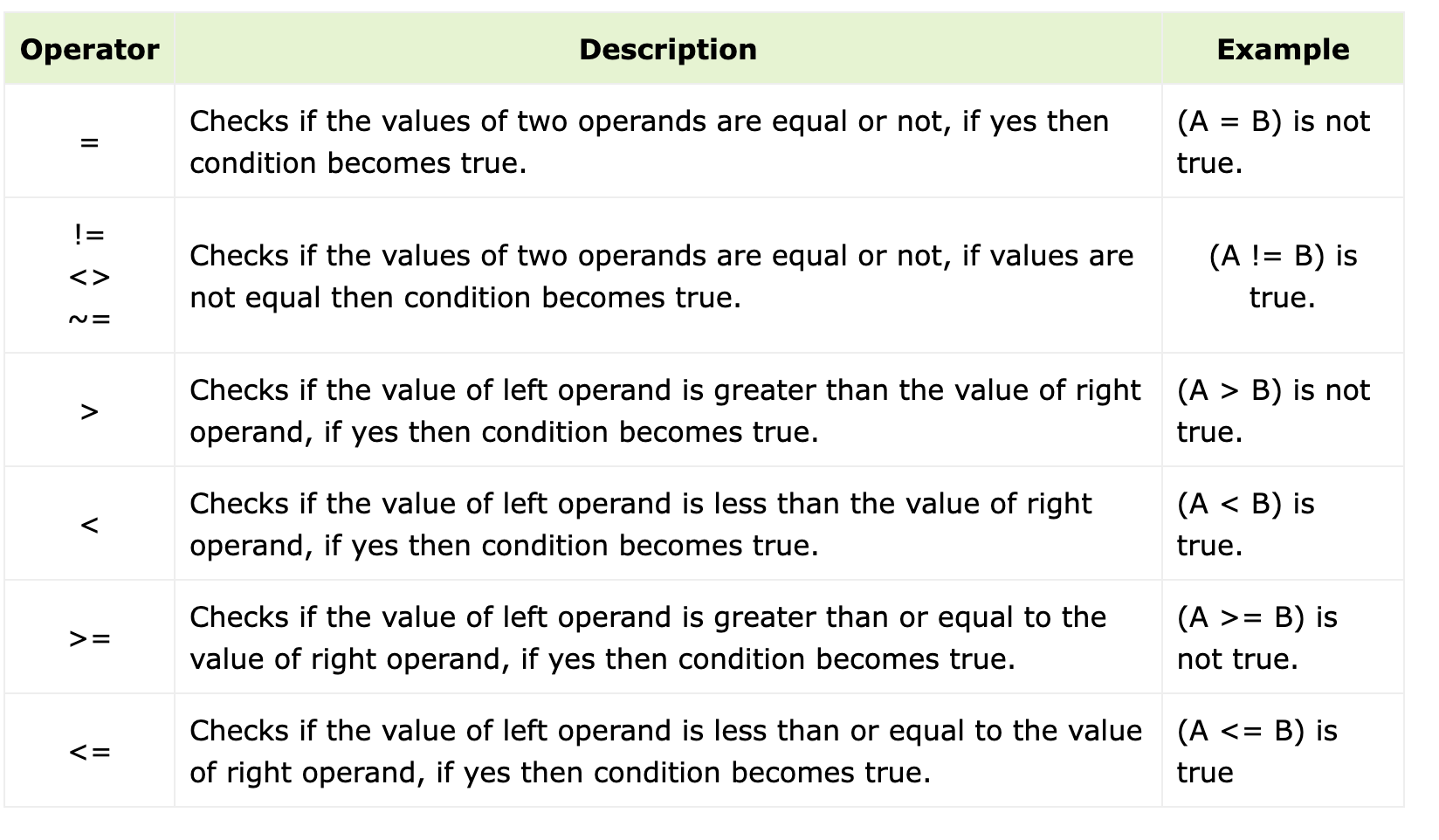
A **composite** type has internal components that can be manipulated individually, such as the elements of an array.

A **reference** type holds values, called pointers, that designate other program items.

A **LOB** type holds values, called lob locators, that specify the location of large objects, such as text blocks or graphic images, that are stored separately from other database data.



|  |  |
| --- | --- |
| DECLARE   num1 INTEGER;   num2 REAL;   num3 DOUBLE PRECISION;  BEGIN   null;  END; | DECLARE   SUBTYPE name IS char(20);   SUBTYPE message IS varchar2(100);   salutation name;   greetings message;  BEGIN   salutation := 'Reader ';   greetings := 'Welcome to the World of PL/SQL';   dbms\_output.put\_line('Hello ' || salutation || greetings);  END; |



|  |  |
| --- | --- |
| name varchar2(25); | DECLARE   a integer := 10;   c integer;  BEGIN   c := a + b;   dbms\_output.put\_line('Value of c: ' || c); END; |
| address varchar2(100); |
| greetings varchar2(20) DEFAULT 'Have a Good Day'; |
| counter binary\_integer := 0; |
| sales number(10, 2); |
| PI CONSTANT NUMBER := 3.141592654; |

**DYNAMODB**

Amazon DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability

|  |  |  |
| --- | --- | --- |
|  | **Explanation** | **Example** |
| **Table** | Like SQL | Users |
| **Item** | Group of Attributes | A current user |
| **Attribute** | Fundament Data Element | userID |

|  |  |
| --- | --- |
| **Primary Key** | Uniquely identify an Item uses internal hash function |
| **Partition Key** | Simple Primary Key (One attribute) |
| **Composite Primary Key** | Partition key and sort key |

|  |  |
| --- | --- |
| **PutItem** | Create an item |
| **GetItem** | Read an item |
| **UpdateItem** | Update an item |
| **DeleteItem** | Delete an item |
| **BatchGetItem** | Read up to 100 items from one or more tables |
| **BatchWriteItem** | Create or delete up to 25 items in one or more tables |

|  |  |
| --- | --- |
| **Scalar Types** | A scalar type can represent exactly one value |
|  | number, string, binary, Boolean, and null |
| **Document Types** | A document type can represent a complex structure with nested attributes like JSON |
|  | The document types are list and map |
| **Set Types** | A set can be multiple scalar values. |
|  | The set types are string set, number set, and binary set. |