# Day 02 SQL ( Rules, Data Types, Constraints )

## The 12 Rules of E.F Codd

If we want to store data into tables on that time we should follow standard rules of EF Codd, those are listed below :

### Rule of Information (Rule 1)

- All information in a relational database is represented explicitly at the logical level as atomic values in tables.
- Data must be stored in rows and columns (tables) and not as pointers or hierarchical structures.

### Rule of Guaranteed Access (Rule 2)

- Every piece of data (value) in a database is accessible by specifying a combination of table name, primary key, and column name.
- There are no hidden data points or proprietary access methods.

### Rule of Systematic Null Value Handling (Rule 3)

- Null values (representing missing or inapplicable information) must be uniformly supported and distinct from zero, blanks, or other default values.
- Null values should indicate the absence of data and not a specific value.

### Rule of Dynamic Online Catalog (Rule 4)

- The database's structure (metadata) must be stored in tables and accessible like regular data.
- Metadata must support the same querying and updating mechanisms as user data.

**Rule of Comprehensive Data Language ( Rule 5 )**

- Use one consistent language (like SQL) for everything: querying, updating, and managing the database.
- The same language should handle security and rules too.

**Rule of View Updating ( Rule 6 )**

- Changes made through a view should update the original data.
- Views (customized presentations of data) should support updates seamlessly.

**Rules of High-Level Insert, Delete, Update ( Rule 7 )**

- Simple Term: Perform operations on sets of data, not just individual rows.
- Explanation: You can manipulate multiple rows in one go instead of one at a time.

**Rule of Physical Data Independence ( Rule 8 )**

- Changes in storage shouldn't affect how you access data.
- The database should work the same, even if its underlying storage changes.

**Rule of Logical Data Independence ( Rule 9 )**

- Changing the table structure shouldn't affect existing applications.
- You can modify the database schema without rewriting your programs.

**Rule of integrity Independence ( Rule 10 )**

- Data integrity rules must be part of the database.
- Constraints like "a column must not be empty" are stored in the database, not in programs.

**Rule of Distribution Independence ( Rule 11 )**

- It shouldn't matter where the data is stored.
- Whether data is on one server or many, it should behave as if it's in one place. ( This is applicable to tables as well )

**Rule of Non-subversion Rule ( Rule 12 )**

- Simple Term: The database must only be accessible through its defined rules and language.
- Explanation: You shouldn't bypass the database's rules using shortcuts or external tools.

# DATA TYPES

**Data Type :**

- Data type is used to define what kind of data can be stored in each column of a table.
- Below are the most commonly used Oracle SQL data types.
  - Character Data Types
  - Numeric Data Types
  - Date and Time Data Types
  - Binary Data Types
  - Miscellaneous Data Types

**Character Data Types :**

Character data types are used to store text-based values:

**CHAR**:

A fixed-length string of characters. If the string is shorter than the defined length, it is padded with spaces.

```
CREATE TABLE countries (
        country_code CHAR(3),
        country_name VARCHAR2(100)
);
```

## VARCHAR & VARCHAR2

### VARCHAR:

VARCHAR is a standard SQL datatype used to store variable-length strings.

- Oracle: In Oracle, VARCHAR is effectively the same as VARCHAR2 (at least up until Oracle 12c), but it's not recommended to use VARCHAR because it may be deprecated in future releases of Oracle.

### VARCHAR2:

VARCHAR2 is an Oracle-specific datatype for variable-length strings.

- VARCHAR size (1,2000)
- VARCHAR2 size ( 1,4000)

### NVARCHAR2:

NVARCHAR2 stores variable-length Unicode characters, meaning it only uses as much space as required to store the string,

```
CREATE TABLE wine_bottle (
   wine_id NUMBER PRIMARY KEY,
   wine_name NVARCHAR2(100),
   wine_type NCHAR(10),
   region NVARCHAR2(50),
   vintage_year NUMBER,
   description NVARCHAR2(200)
);
```

**CLOB** :

A large character object used to store long texts (up to 4GB).

```
CREATE TABLE articles (
        article_id NUMBER PRIMARY KEY,
        title NVARCHAR2(100),
        author NVARCHAR2(50),
        publication_date DATE,
        content CLOB
    );
```

**NUMERIC DATA TYPE:**

- **NUMBER**: Stores fixed and floating-point numbers. It can be defined with precision (total digits) and scale (digits after the decimal point).
- **INTEGER**: A subtype of NUMBER, used for whole numbers (no decimals).
- **FLOAT**: A subtype of NUMBER, used for floating-point numbers.
- **DECIMAL**: Equivalent to NUMBER, but with a focus on fixed-point numbers.

**NUMBER: NUMBER( <precision>, [scale] );**

The NUMBER data type stores both fixed and floating-point numbers. It can be defined with precision (total digits) and scale (digits after the decimal point).

```
CREATE TABLE employees (
employee_id NUMBER(6),
salary NUMBER(8, 2)
);
```

**INTEGER**:

INTEGER is a subtype of the NUMBER data type and is used to store whole numbers (i.e., no decimal points).

```
CREATE TABLE orders (
    order_id INTEGER,
   quantity INTEGER
);
```

**FLOAT:**

FLOAT is a subtype of the NUMBER data type and used to store floating point numbers ( with fractional part too )

```
CREATE TABLE temperature_readings (
reading_id INTEGER,
temperature FLOAT
);
```

**DECUMAL**

The **DECIMAL** data type is functionally equivalent to the **NUMBER** data type in Oracle, but it is generally used to store fixed-point numbers, meaning that the number of digits after the decimal point is fixed.

```
CREATE TABLE financial_records (
    transaction_id INTEGER,
    amount DECIMAL(10, 2)
);
```

**DATE AND TIME Data Types :**

- DATE
- TIMESTAMP
- TIMESTAMP WITH TIME ZONE
- TIMESTAMP WITH LOCAL TIME ZONE
- INTERVAL

DATE :

- Stores date and time values with a precision of seconds. The DATE data type includes both the
- date and time components, and it stores values in the format DD-MON-YYYY HH24:MI:SS.
- Example:

```
CREATE TABLE events (
    event_id NUMBER PRIMARY KEY,
    event_name VARCHAR2(100),
    event_date DATE
);
```

TIMESTAMP

- Extends the DATE data type to store fractional seconds, providing greater precision for date and time values.
- The TIMESTAMP format includes seconds, with optional fractions of a second.

- Example:

```
CREATE TABLE transactions (
transaction_id NUMBER PRIMARY KEY,
transaction_time TIMESTAMP
);
```

## TIMESTAMP WITH TIME ZONE

- Stores a TIMESTAMP with an associated time zone. This data type is useful for recording timestamps that need to be adjusted to specific time zones.
- Example:

```
CREATE TABLE global_events (
    event_id NUMBER PRIMARY KEY,
    event_time TIMESTAMP WITH TIME ZONE
);
```

## TIMESTAMP WITH LOCAL TIME ZONE

- Stores a TIMESTAMP in the local time zone, meaning the time zone information is automatically adjusted when stored and retrieved.
- This is useful for ensuring that time data is consistent across multiple regions.
- Example

```
CREATE TABLE user_activity (
   user_id NUMBER PRIMARY KEY,
   login_time TIMESTAMP WITH LOCAL TIME ZONE
);
```

## INTERVAL

- Represents a time interval, which can be specified in terms of days, hours, minutes, seconds, and fractions of seconds.
- The INTERVAL data type allows you to store durations (e.g., INTERVAL '1' DAY or INTERVAL '5' HOUR TO MINUTE).
- Example

```
CREATE TABLE work_shifts (
   shift_id NUMBER PRIMARY KEY,
   shift_duration INTERVAL DAY TO SECOND
   );
```

## BINARY DATA TYPES:

Binary data types are used to store binary data (such as images or files):

**BLOB**: A large binary object used to store large binary data (up to 4GB).
**BINARY_FLOAT** : A 32-bit floating-point number.
**BINARY_DOUBLE** : A 64-bit floating-point number.
**RAW** : A variable-length binary string.
**LONG RAW** : A longer version of RAW.

BLOB:

- A BLOB is used to store large binary data, such as images, audio, or video files, with a storage capacity of up to 4GB.
- It is suitable for handling large multimedia files that require binary storage.

```
CREATE TABLE multimedia_files (
        file_id NUMBER PRIMARY KEY,
        file_data BLOB
 );
```

BINARY_FLOAT

- The BINARY_FLOAT data type is a 32-bit floating-point number.
- It is used to store single-precision floating-point values, offering a compact storage option for real numbers.
- Example :

```
CREATE TABLE product_measurements (
        product_id NUMBER PRIMARY KEY,
        weight BINARY_FLOAT
);
```

BINARY_DOUBLE

- The BINARY_DOUBLE data type is a 64-bit floating-point number.
- It stores double-precision floating-point numbers and is suitable for more precise calculations involving real numbers.
- Example:

```
CREATE TABLE temperature_readings (
        reading_id NUMBER PRIMARY KEY,
        temperature BINARY_DOUBLE
);
```

RAW

- The RAW data type is used to store variable-length binary data, such as encrypted data, or binary values like GUIDs.
- It is often used when you need to store raw data without interpretation.
- Example :

```
CREATE TABLE sensitive_data (
        record_id NUMBER PRIMARY KEY,
        data RAW(100)
);
```

LONG RAW

- The LONG RAW data type is similar to RAW, but it allows for storing larger binary data.
- It is used to store binary data exceeding the length limitations of the RAW data type.
- Example:

```
CREATE TABLE audio_files (
    file_id NUMBER PRIMARY KEY,
    audio_data LONG RAW
);
```

**Miscellaneous Data Types**

- ROWID
- UROWID
- XMLType
- Object Types

ROWID :

- A unique identifier for a row in a table. The ROWID data type represents the physical location of a row in the database. Each row in a table has a unique ROWID.
- ROWID is a default pseudo column for every table, which provides the physical location of a row in the database.

```
SELECT rowid, employee_name FROM employees;
```

UROWID :

- UROWID stands for "Universal ROWID".
- It is used to represent the ROWID in a format that is not tied to the physical structure of a specific Oracle database.
- It is useful when working with external tables, remote databases, or database links.
- UROWID can be used in situations where the physical storage details are irrelevant or may change across different databases.

```
SELECT UROWID, emp_id, emp_name
FROM employees@remote_db_link
WHERE emp_id = 101;
```

XMLType:

- XMLType is a special Oracle data type used to store XML data in a structured way.
- It provides storage and manipulation capabilities for XML data. XMLType can store XML documents or fragments and allows you to perform XML-specific operations, such as querying and transforming XML using SQL/XML functions.
- Storage: XMLType data can be stored either as CLOB (Character Large Object) or as an object-relational type in the database.

```
CREATE TABLE empxml (
        emp_id NUMBER,
        emp_name VARCHAR2(100),
        emp_data XMLType
);
INSERT INTO empxml (emp_id, emp_name, emp_data)
VALUES (101, 'John Doe',
XMLType('<employee><emp_id>101</emp_id><emp_name>John
Doe</emp_name><salary>50000</salary></employee>'));
```

**Object Types :**

- Object Types in Oracle represent user-defined data types that encapsulate both data and methods (procedures and functions) that can operate on that data.
- These types are modeled on the concept of objects in object-oriented programming and are used to represent real-world entities more naturally.
- Object types can have attributes (fields) and methods procedures/functions).

Example :

```
CREATE TYPE Employee AS OBJECT (
        emp_id NUMBER,
        emp_name VARCHAR2(100),
        salary NUMBER,
        MEMBER FUNCTION get_full_name RETURN
       VARCHAR2
);
```

Now, create the body of the object type:

```
CREATE TYPE BODY Employee AS
  MEMBER FUNCTION get_full_name RETURN VARCHAR2 IS
  BEGIN
    RETURN 'Mr./Ms. ' || emp_name;
  END;
END;
```

We can create a table based on this object type :

```
CREATE TABLE employees_of_type OF Employee;
```

Now, Inserting data into table

```
INSERT INTO employees_of_type VALUES (Employee(101, 'John
Doe', 50000));
```

Calling the method to get full name

```
SELECT emp_name.get_full_name() FROM employees_of_type
WHERE emp_id = 101;
```

# CONSTRAINTS IN SQL:

- **NOT NULL**
- **UNIQUE**
- **PRIMARY KEY**
- **FOREIGN KEY**
- **CHECK**
- **DEFAULT**
- **REFERENCES**
- **INDEX** (Though not strictly a constraint, it is often used in relation to constraints)
- **EXCLUDE** (for Spatial data types)

## NOT NULL :

The NOT NULL constraint ensures that a column cannot store NULL values. When applied to a column, it enforces that every row in the table must have a value for that column. If an attempt is made to insert or update a record without providing a value for a column defined as NOT NULL, Oracle will raise an error.

```
CREATE TABLE employees (
employee_id NUMBER PRIMARY KEY,
first_name VARCHAR2(50) NOT NULL,
last_name VARCHAR2(50) NOT NULL,
email VARCHAR2(100) NOT NULL
);
```

Strange Cases and Common Mistakes :

- Inserting NULL  with NOT NULL columns
- In some cases, developers mistakenly treat an empty string ('') as NULL. In Oracle, an empty string is stored as NULL for VARCHAR and CHAR types, but NOT NULL would still prevent that.
- Implicit NOT NULL on Primary Key and Unique Columns
- Handling NOT NULL Constraints During Table Alterations

**UNIQUE :**

- UNIQUE constraint ensures that all values in a column or a set of columns are distinct, preventing duplicate entries.
- While it's a fundamental concept in database design, there are some nuances and common mistakes that developers might encounter when using UNIQUE.
- The UNIQUE constraint can be applied either at the column level or the table level.
- Column Level Example :

```
CREATE TABLE employees (
    emp_id INT PRIMARY KEY,
    emp_name VARCHAR2(100),
    emp_email VARCHAR2(100) UNIQUE  -- Applying
UNIQUE constraint
);
```

- Table Level Constraint :

```
CREATE TABLE employees (
    emp_id INT PRIMARY KEY,
    emp_name VARCHAR2(100),
    emp_email VARCHAR2(100),
    CONSTRAINT unique_email UNIQUE (emp_email)
);
```

**Composite UNIQUE constraint (Multiple columns):**

You can apply a UNIQUE constraint to a combination of columns. This ensures that the combination of values in those columns is unique.

```
CREATE TABLE order_details (
    order_id INT,
    product_id INT,
    quantity INT,
    CONSTRAINT unique_order_product UNIQUE (order_id, product_id)
);
```

**Strange Situations :**

NULL Values and UNIQUE Constraints :

- In Oracle, UNIQUE constraints allow multiple NULL values in a column because NULL is treated as a "non-value" and Oracle considers multiple NULLs as not equal to each other.

Incorrect Use of UNIQUE with PRIMARY KEY:

- PRIMARY KEY constraint inherently implies UNIQUE and NOT NULL.
- Applying a UNIQUE constraint to a column that already has a PRIMARY KEY constraint is unnecessary and redundant.
- CREATE TABLE employees (
  emp_id INT PRIMARY KEY,
  emp_name VARCHAR2(100),
  emp_email VARCHAR2(100) UNIQUE
  );

**FOREIGN KEY CONSTRAINT:**

- A foreign key constraint is a rule that maintains referential integrity between two tables in a relational database.
- It ensures that the value in one table's column (the foreign key) corresponds to a valid, existing value in another table's column (the primary key or a unique key).
- Ensures that relationships between tables remain consistent.
- Prevents orphaned records, i.e., records in the child table that don’t have a matching record in the parent table.

- Purpose:
  - Ensures that relationships between tables remain consistent.
  - Prevents orphaned records, i.e., records in the child table that don't have a matching record in the parent table.
- How it works:
  - The foreign key in the child table points to the primary key in the parent table.
  - For every record in the child table, there must be a corresponding record in the parent table, unless the foreign key allows NULL values.
- **Actions on Deletion/Update:**
  - CASCADE: When the parent record is deleted or updated, the change is automatically reflected in the child table.
  - SET NULL: When the parent record is deleted or updated, the foreign key in the child table is set to NULL.
  - RESTRICT: Prevents the deletion or update of the parent record if it is referenced by a foreign key in the child table.
  - NO ACTION: Same as RESTRICT, but the check is deferred until the end of the transaction.

Syntax :

FOREIGN KEY (column_name) REFERENCES parent_table (parent_column)
ON DELETE <TYPE_OF_ACTION>
ON UPDATE < TYPE_OF_ACTION >

( we can set on delete null or on update null )

**DEFAULT CONSTRAINT :**

- The DEFAULT constraint is used in SQL to provide a default value for a column when no value is specified during an insert operation.
- If no value is provided for that column, the database automatically inserts the default value.

- CREATE TABLE Employees (
  EmployeeID INT PRIMARY KEY,
  Name VARCHAR(100),
  JoiningDate DATE DEFAULT CURRENT_DATE
  );

**REFERENCES CONSTRAINT :**

- The REFERENCES constraint is used to create a foreign key relationship between two tables.
- It links a column in one table to the primary key (or unique key) of another table, ensuring referential integrity.
- This means that a value in the referencing table (child) must exist in the referenced table (parent).

  CREATE TABLE Orders (
  OrderID INT PRIMARY KEY,
  EmployeeID INT,
  OrderDate DATE,
  FOREIGN KEY (EmployeeID) REFERENCES
  Employees(EmployeeID)
  );

(

**NOTE**:
ECLUDE IS NOT AVIABLE DIRECLTY IN ORACLE SQL.
BUT USING TRIGGERS CONCEPT WE CAN CREATE
EXCLUDE FUNCTIONALITY EXPLAICITLY

**NOTE:**
INDEX CONCEPT WILL BE DISCUSS IN UP COMING PDFS

)