# SQL Interview Notes

## ❓ What is the difference between NVARCHAR and VARCHAR?

**Answer:**  
- VARCHAR stores **non-Unicode** characters (1 byte per character).  
- NVARCHAR stores **Unicode** characters (2 bytes per character).

**Explanation:**  
- Use VARCHAR when storing English or ASCII text → less storage.  
- Use NVARCHAR for multilingual data (Hindi, Chinese, Arabic, etc.).

**Example:**

-- VARCHAR example (English only)  
DECLARE @nameV VARCHAR(20) = 'Krish';  
  
-- NVARCHAR example (Unicode, supports any language)  
DECLARE @nameN NVARCHAR(20) = N'कृष';

## ❓ What is a Stored Procedure?

**Answer:**  
A **Stored Procedure** is a precompiled collection of one or more SQL statements stored in the database that can be executed as a single unit.

**Explanation:**  
- Created once and executed multiple times.  
- Improves performance (cached execution plan).  
- Adds reusability, modularity, and security.

**Example:**

CREATE PROCEDURE GetCustomerById  
 @CustomerId INT  
AS  
BEGIN  
 SELECT \* FROM Customers WHERE Id = @CustomerId;  
END;  
  
EXEC GetCustomerById @CustomerId = 5;

**Advantages:** Performance, Security, Reusability, Maintainability.

## ❓ What is SQL Injection and how to prevent it?

**Answer:**  
**SQL Injection** is a vulnerability where attackers inject malicious SQL into input fields to gain unauthorized access to a database.

**Vulnerable Example:**

string query = "SELECT \* FROM Users WHERE Username = '" + username +   
 "' AND Password = '" + password + "'";

👉 Attacker inputs: admin' OR '1'='1 → bypass login.

**Prevention:**  
1. **Parameterized Queries / Prepared Statements**

SqlCommand cmd = new SqlCommand("SELECT \* FROM Users WHERE Username=@u AND Password=@p", conn);  
cmd.Parameters.AddWithValue("@u", username);  
cmd.Parameters.AddWithValue("@p", password);

1. **Stored Procedures**
2. **Input Validation & Sanitization**
3. **Use ORM frameworks** (e.g., Entity Framework)
4. **Least Privilege Principle** for DB accounts.

## ❓ What is the use of a Trigger?

**Answer:**  
A **Trigger** is a special stored procedure that executes automatically in response to events (INSERT, UPDATE, DELETE) on a table.

**Example:**

CREATE TRIGGER trg\_AfterInsert  
ON Employees  
AFTER INSERT  
AS  
BEGIN  
 INSERT INTO AuditLog (Action, ActionDate)  
 VALUES ('New Employee Inserted', GETDATE());  
END;

**Use Cases:** Audit logs, enforcing rules, cascading changes.

## ❓ What is the use of a Cursor?

**Answer:**  
A **Cursor** allows row-by-row processing of query results.

**Example:**

DECLARE @name NVARCHAR(50);  
DECLARE employee\_cursor CURSOR FOR SELECT Name FROM Employees;  
OPEN employee\_cursor;  
FETCH NEXT FROM employee\_cursor INTO @name;  
WHILE @@FETCH\_STATUS = 0  
BEGIN  
 PRINT 'Employee: ' + @name;  
 FETCH NEXT FROM employee\_cursor INTO @name;  
END;  
CLOSE employee\_cursor;  
DEALLOCATE employee\_cursor;

**Use Cases:** When row-wise processing is necessary (though less efficient than set-based queries).

## ❓ What is Normalization?

**Answer:**  
Normalization organizes data to reduce redundancy and improve data integrity. It splits large tables into smaller related ones, preventing anomalies in Insert, Update, and Delete operations.

### 📌 Types of Normal Forms (with Examples)

| Normal Form | Focus Area | Prevents | Example (Before) | Example (After) | Explanation |
| --- | --- | --- | --- | --- | --- |
| **1NF** | Atomic values | Repeating groups | Students(ID, Name, Phones) → 1, Krish, 98765, 99887 | Students(ID, Name, Phone) → 1, Krish, 98765 ; 1, Krish, 99887 | No multiple values in a single column. |
| **2NF** | Full dependency on key | Partial dependency | Orders(OrderID, ProductID, ProductName, Qty) | Split into Orders(OrderID, ProductID, Qty) & Products(ProductID, ProductName) | Non-key columns must depend on the full primary key. |
| **3NF** | No transitive dependency | Derived dependency | Students(ID, Name, DeptName, DeptHOD) | Students(ID, Name, DeptName) & Departments(DeptName, DeptHOD) | Non-key columns shouldn’t depend on other non-key columns. |
| **BCNF** | Stronger 3NF | Anomalies from non-candidate keys | Courses(Course, Teacher, Room) | CourseTeacher(Course, Teacher) & TeacherRoom(Teacher, Room) | Every determinant must be a candidate key. |
| **4NF** | No multi-valued dependency | Independent sets of values | StudentID, Hobby, Language → 1, Football, Hindi / 1, Painting, English | StudentHobby(StudentID, Hobby) & StudentLanguage(StudentID, Language) | Prevents storing unrelated attributes in the same table. |
| **5NF** | No join dependency | Redundancy in complex joins | Course, Teacher, Book with repeated rows | Split into CourseTeacher, CourseBook, TeacherBook | Eliminates redundancy caused by join dependencies. |

## 🎯 Why Normalization?

* Removes redundancy
* Improves data integrity
* Makes maintenance easier
* Reduces anomalies in database operations