Advanced SQL Topics Overview

Subqueries

A query nested inside another query. Example:

```
SELECT * FROM employees WHERE department_id = (SELECT department_id FROM departments WHERE department_name = 'Sales');
```

Window Functions

Perform calculations across a set of table rows related to the current row. Example:

```
SELECT employee_id, salary, AVG(salary) OVER (PARTITION BY department_id) AS avg_salary FROM employees;
```

Common Table Expressions (CTE)

Temporary result set defined within the execution scope of a SELECT, INSERT, UPDATE, or DELETE statement. Example:

```
WITH Sales_CTE (salesperson, total_sales) AS (
    SELECT salesperson, SUM(sales) FROM sales GROUP BY salesperson
)
SELECT * FROM Sales_CTE WHERE total_sales > 1000;
```

Stored Procedures

A prepared SQL code that you can save and reuse. Example:

```
CREATE PROCEDURE GetEmployeeCount @DepartmentID INT
AS
BEGIN
    SELECT COUNT(*) FROM employees WHERE department_id = @DepartmentID;
END;
```

Triggers

Automatically execute a response to certain events on a particular table or view. Example:

```
CREATE TRIGGER trgAfterInsert ON employees

FOR INSERT

AS

BEGIN

    PRINT 'An insert event has occurred';

END;
```

Indexes

Improve the speed of data retrieval operations on a database table. Example:

```
CREATE INDEX idx_employee_name ON employees (last_name);
```

Transactions

A sequence of operations performed as a single logical unit of work. Example:

```
BEGIN TRANSACTION;

UPDATE account SET balance = balance - 100 WHERE account_id = 1;

UPDATE account SET balance = balance + 100 WHERE account_id = 2;

COMMIT;
```

Views

A virtual table based on the result set of an SQL statement. Example:

```
CREATE VIEW Sales_View AS

SELECT salesperson, SUM(sales) AS total_sales FROM sales GROUP BY salesperson;
```

Normalization

Organizing data to reduce redundancy and improve data integrity. Example:

- First Normal Form (1NF): Ensure each column contains atomic values.

- Second Normal Form (2NF): Remove partial dependencies.
- Third Normal Form (3NF): Remove transitive dependencies.

Partitioning

Dividing a database table into smaller, more manageable pieces. Example:

```
CREATE TABLE orders_partitioned (
    order_id INT,
    order_date DATE,
    customer_id INT
)
PARTITION BY RANGE (order_date) (
    PARTITION p0 VALUES LESS THAN ('2022-01-01'),
    PARTITION p1 VALUES LESS THAN ('2023-01-01')
);
```

Advanced JOINs

Combining rows from multiple tables with complex conditions. Example:

```
SELECT e.employee_id, e.last_name, d.department_name
FROM employees e
FULL OUTER JOIN departments d ON e.department_id = d.department_id;
```

Full-Text Search

Searching for text in large datasets using full-text indexes. Example:

```
CREATE FULLTEXT INDEX ON documents (document_text)
KEY INDEX PK_Documents;
```

Recursive Queries

A CTE that references itself to process hierarchical data. Example:

```
WITH RECURSIVE EmployeeCTE AS (
    SELECT employee_id, manager_id, employee_name FROM employees WHERE manager_id IS
NULL
   UNION ALL
   SELECT e.employee_id, e.manager_id, e.employee_name
   FROM employees e
   INNER JOIN EmployeeCTE ecte ON e.manager_id = ecte.employee_id
)
SELECT * FROM EmployeeCTE;
```

Dynamic SQL

Constructing SQL statements dynamically at runtime. Example:

```
DECLARE @sql NVARCHAR(MAX);
SET @sql = 'SELECT * FROM ' + @table_name;
EXEC sp_executesql @sql;
```

Error Handling

Managing errors using TRY...CATCH blocks. Example:

```
BEGIN TRY
   -- Generate a divide-by-zero error
   SELECT 1 / 0;
END TRY
BEGIN CATCH
   SELECT ERROR_MESSAGE() AS ErrorMessage;
END CATCH;
```

Performance Tuning

Techniques to improve query performance. Example:

- Use indexes appropriately.
- Avoid SELECT *.
- Use EXISTS instead of IN for subqueries.