Optimize SQL Queries for Performance

Gerald Britton

Pluralsight Author

@GeraldBritton www.linkedin.com/in/geraldbritton

Database Factors Affecting Performance

CPU utilization

RAM capacity

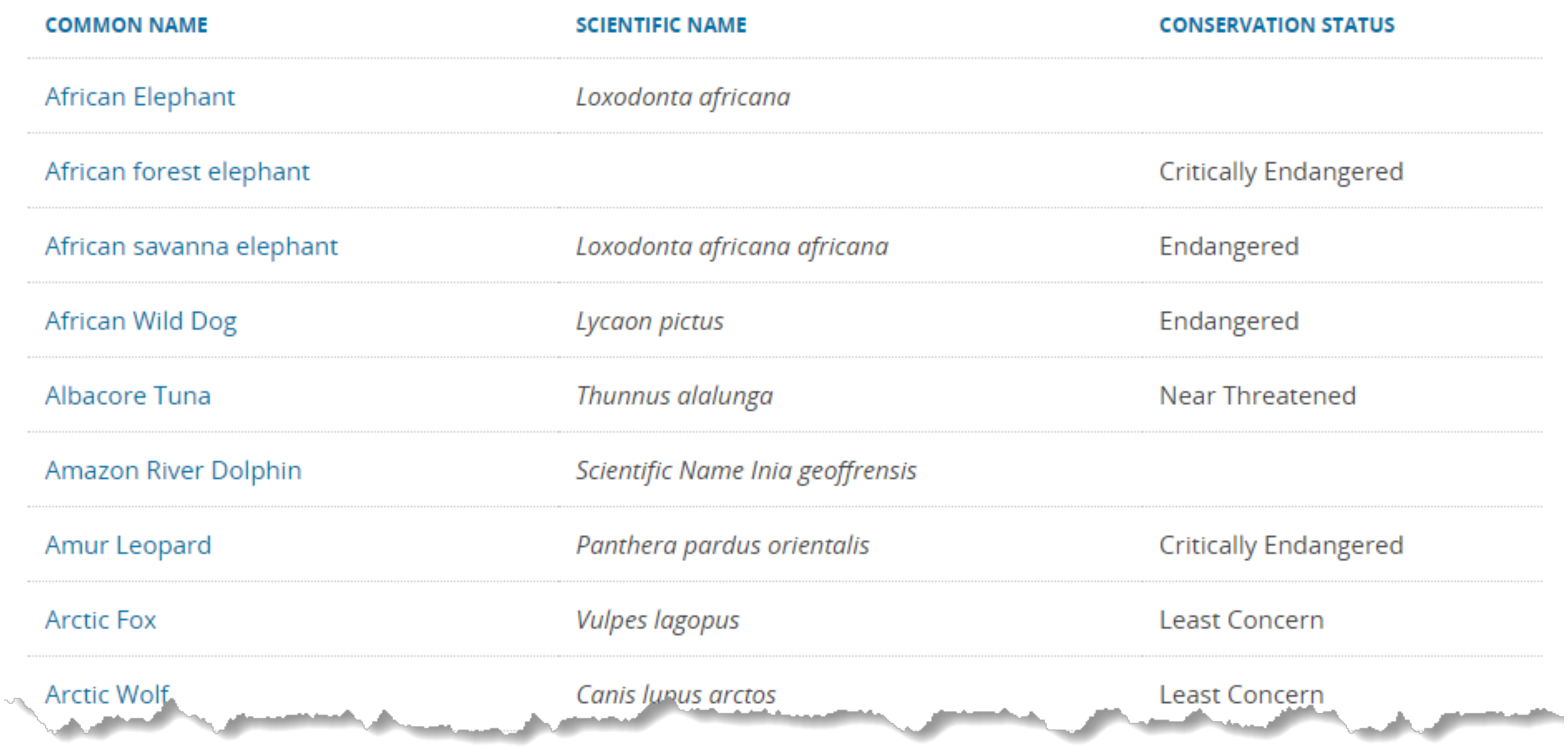
I/O throughput

Network latency

Indexing for Performance

Create and use table indexing to support your query

What Is an Index?



COMMON NAME	SCIENTIFIC NAME	CONSERVATION STATUS
African Elephant	*Loxodonta africana*	
African forest elephant		Critically Endangered
African savanna elephant	*Loxodonta africana africana*	Endangered
African Wild Dog	*Lycaon pictus*	Endangered
Albacore Tuna	*Thunnus alalunga*	Near Threatened
Amazon River Dolphin	*Scientific Name Inia geoffrensis*	
Amur Leopard	*Panthera pardus orientalis*	Critically Endangered
Arctic Fox	*Vulpes lagopus*	Least Concern
Arctic Wolf	*Canis lupus arctos*	Least Concern

https://www.worldwildlife.org/species/directory

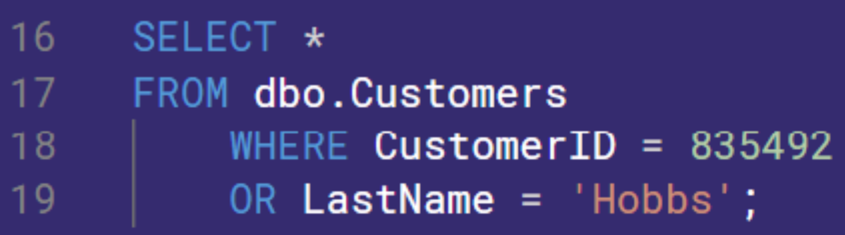
```sql
CREATE TABLE dbo.Customers(
    CustomerID int,
    FirstName varchar(255),
    LastName varchar(255),
    Street varchar(255),
    StreetNumber char(10),
    Unit char(10),
    City varchar(255),
    StateProvince varchar(255),
    ISO3_Country char(3),
    EmailAddress varchar(254),
    HomePhone numeric(15, 0),
    MobilePhone numeric(15, 0),
);
```

◄ **Customers table**

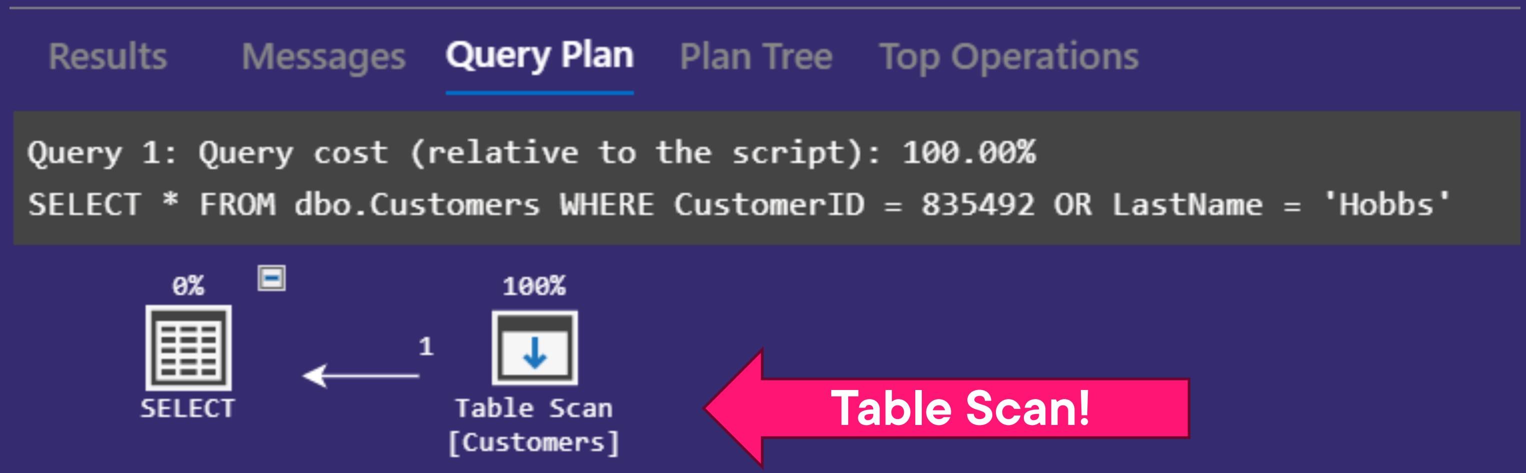
```sql
SELECT *
FROM dbo.Customers
    WHERE CustomerID = 835492
    OR LastName = 'Hobbs';
```

◄ **Query to select a particular customer**

Query Plan Showing Potential Problem



```sql
SELECT *
FROM dbo.Customers
    WHERE CustomerID = 835492
    OR LastName = 'Hobbs';
```



```
Query 1: Query cost (relative to the script): 100.00%
SELECT * FROM dbo.Customers WHERE CustomerID = 835492 OR LastName = 'Hobbs'
```

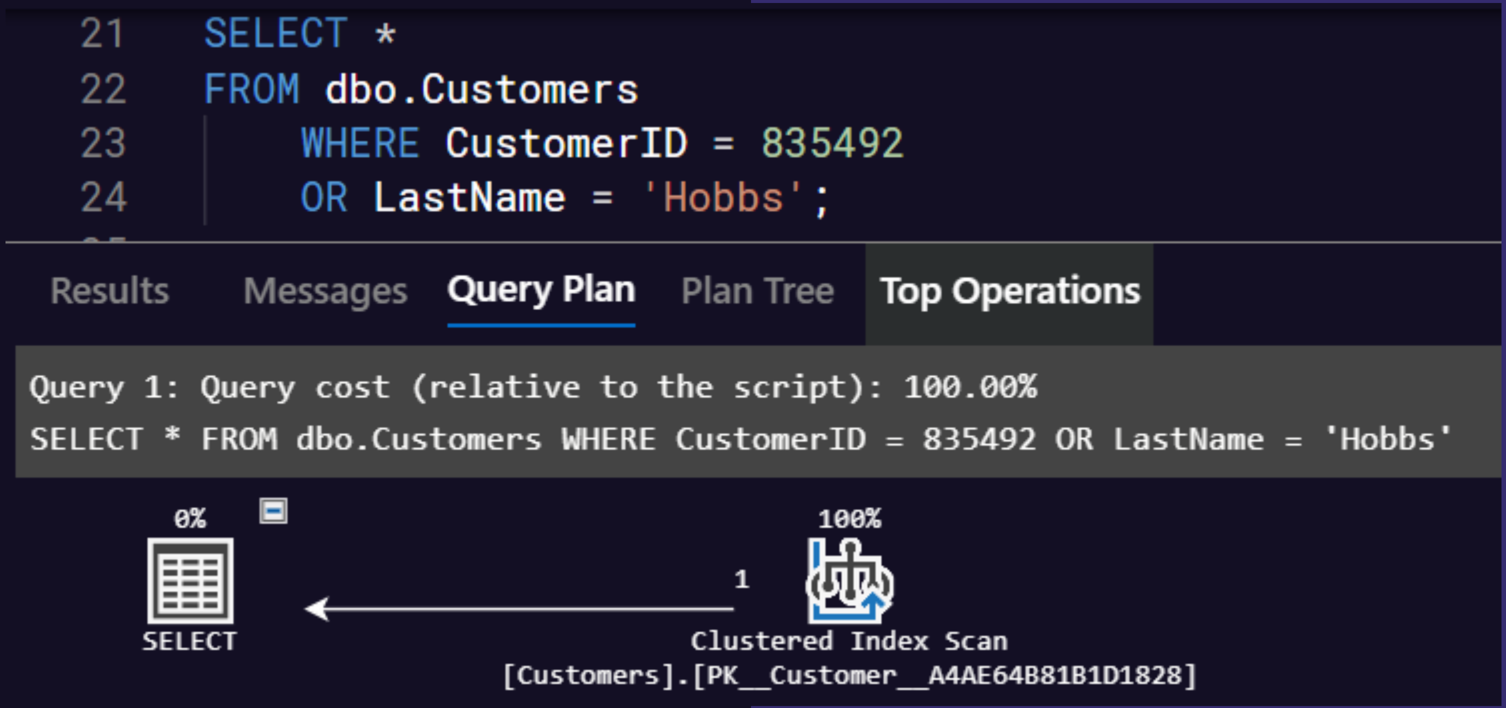
Table Scan!

```
CREATE TABLE dbo.Customers (
        CustomerID INT PRIMARY KEY,
        …
```

- PRIMARY KEY creates an index

```
CREATE INDEX idx_LastName
    ON dbo.Customers (LastName);
```

- Put index on the LastName column

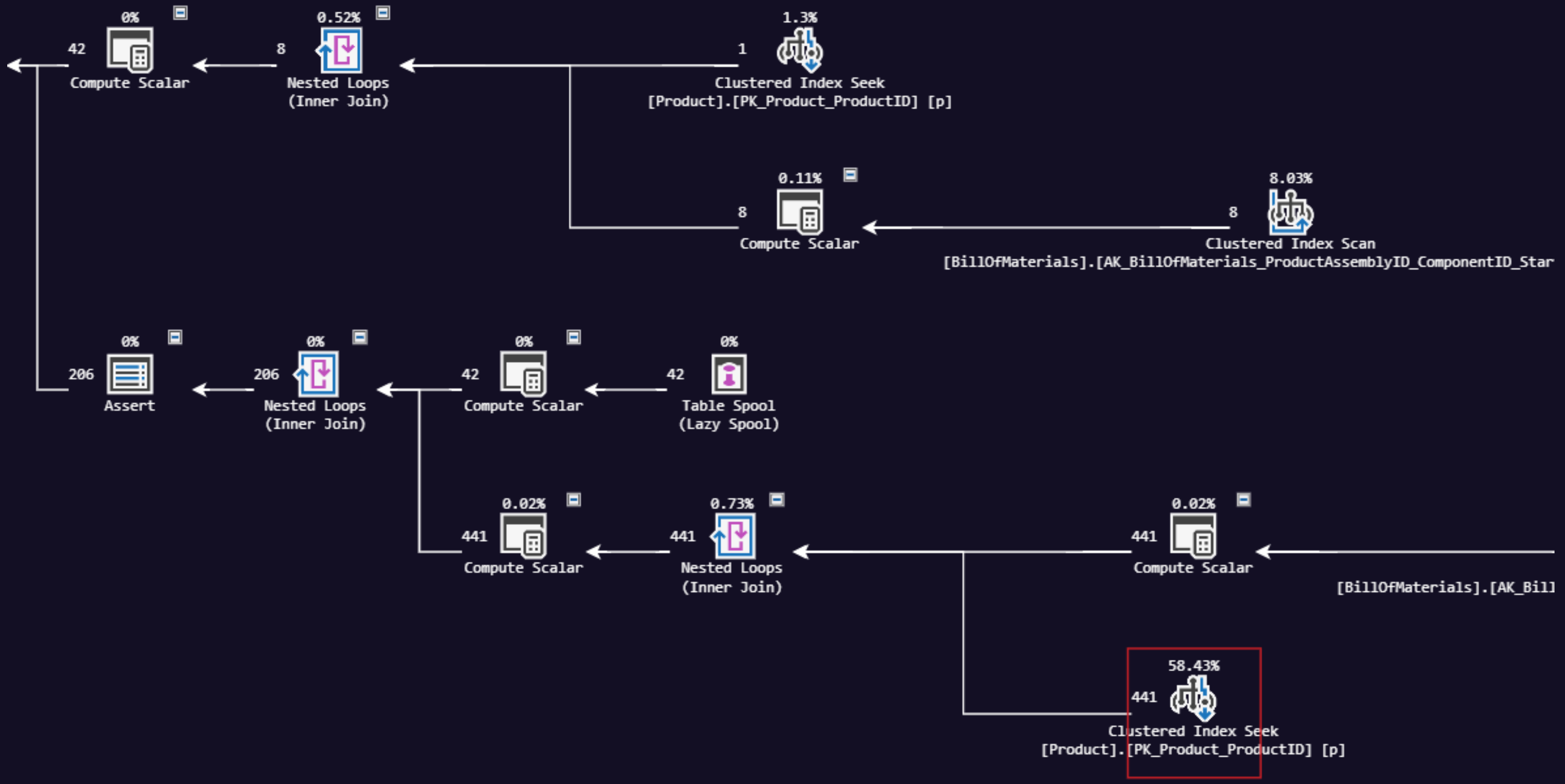


```
SELECT *
FROM dbo.Customers
    WHERE CustomerID = 835492
    OR LastName = 'Hobbs';
```

```
Query 1: Query cost (relative to the script): 100.00%
SELECT * FROM dbo.Customers WHERE CustomerID = 835492 OR LastName = 'Hobbs'
```

- Empty Table
- Statistics

Query Plan with Cost and Row Counts



Query Cost Estimation

Database optimizer finds lowest-cost execution plan

Query Plan Caching

Optimizing performance by avoiding the optimizer

Storing query plans in memory or on disk

Parameterizing queries to increase reuse

Invalidating obsolete and underutilized plans

Database Design

A properly-designed database is a basis for performant queries

Optimizing a Database Schema with Normalization

First try

```
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100) UNIQUE,
    phone_number VARCHAR(20),
    street_address VARCHAR(255),
    city VARCHAR(50),
    state VARCHAR(50),
    postal_code VARCHAR(20),
    date_registered DATE
);
```

Normalized

```
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100) UNIQUE,
    phone_number VARCHAR(20),
    address_id INT,
    date_registered DATE,
    FOREIGN KEY (address_id)
        REFERENCES addresses(address_id)
);

CREATE TABLE addresses (
    address_id INT PRIMARY KEY,
    street_address VARCHAR(255),
    city VARCHAR(50),
    state VARCHAR(50),
    postal_code VARCHAR(20)
);
```

Indexing Tables for Typical Queries

PRIMARY KEY Index

```
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100) UNIQUE,
    phone_number VARCHAR(20),
    address_id INT,
    date_registered DATE,
    FOREIGN KEY (address_id)
        REFERENCES
addresses(address_id)
);
```

Business Key Index

```
SELECT customer_id
FROM customers
WHERE last_name LIKE 'Hobbs%’;

CREATE INDEX idx_last_first_name
    ON customers (last_name,
         first_name);
```

Creating Views for Frequent Operations

Table definitions

```
CREATE TABLE customers (
    customer_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100) UNIQUE,
    phone_number VARCHAR(20),
    address_id INT,
    date_registered DATE,
    FOREIGN KEY (address_id)
        REFERENCES addresses(address_id)
);

CREATE TABLE addresses (
    address_id INT PRIMARY KEY,
    street_address VARCHAR(255),
    city VARCHAR(50),
    state VARCHAR(50),
    postal_code VARCHAR(20)
);
```

Customer details view

```
CREATE VIEW customer_details_view AS
SELECT c.customer_id,
       c.first_name || ' ' || c.last_name
            AS customer_name,
       a.street_address,
       a.city,
       a.state,
       a.postal_code
FROM customers c
JOIN addresses a
    ON c.address_id = a.address_id;

SELECT first_name, last_name, street_address
    FROM customer_details_view
    WHERE customer_id = 83247
```

Parameterizing Views with Functions

View

```
CREATE VIEW customer_details_view AS
SELECT c.customer_id,
       c.first_name || ' ' || c.last_name
            AS customer_name,
       a.street_address,
       a.city,
       a.state,
       a.postal_code
FROM customers c
JOIN addresses a
    ON c.address_id = a.address_id;
```

Function

```
CREATE FUNCTION
    get_customer_details_by_id (@p_customer_id INT)
RETURNS TABLE
AS
RETURN
(
    SELECT CONCAT(c.first_name, ' ', c.last_name)
        AS customer_name,
           a.street_address,
           a.city,
           a.state,
           a.postal_code
    FROM customers c
    JOIN addresses a ON c.address_id = a.address_id
    WHERE c.customer_id = @p_customer_id
);

SELECT first_name, last_name, street_address
FROM get_customer_details_by_id(83247);
```

Writing Performant SQL

Creating queries that perform well, are easy to understand and a pleasure to maintain

Use Table Aliases and Two-Part Naming

Without aliases

```
SELECT a, b, c, d
FROM table1
JOIN table2 ON e = f
JOIN table3 ON g = h
WHERE i = 42 AND j = 24;
```

With aliases

```
SELECT t1.a, t1.b, t2.c, t3.d
FROM table1 t1
JOIN table2 t2
    ON t2.e = t1.f
JOIN table3 t3
    ON t3.g = t2.h
WHERE t2.i = 42 AND t2.j = 24;
```

Use CTEs Instead of Nested Subqueries

Nested subqueries

```
SELECT t1.a, t1.b, t2.c, t3.d
FROM table1 t1
JOIN (
    SELECT t2.c, t2.h, t2.i, t2.j
    FROM table2 t2
    JOIN (
        SELECT t3.d FROM table3 t3
        WHERE t3.g = t2.h
    WHERE t2.e = t2.f
    ) t3;
```

Common Table Expressions

```
WITH
    t1 AS (SELECT t1.a,t1.b,t1.f
          FROM table1 t1),
    t2 AS (SELECT t2.c,t2.h,t2.i,t2.j
        FROM table2 t2
        WHERE t2.i = 42 AND t2.j = 24),
    t3 AS (SELECT t3.d FROM table3 t3)

SELECT t1.a, t1.b, t2.c, t3.d
FROM table1 t1
JOIN table2 t2 ON t2.e = t1.f
JOIN table3 t3 ON t3.g = t2.h;
```

Avoid SELECT *

Bad

```
SELECT *
FROM table1 t1
JOIN table2 t2
    ON t2.e = t1.f
JOIN table3 t3
    ON t3.g = t2.h
WHERE t2.i = 42 AND t2.j = 24;
```

- Performance impact
- Readability
- Hidden bugs

Good

```
SELECT t1.a, t1.b, t2.c, t3.d
FROM table1 t1
JOIN table2 t2
    ON t2.e = t1.f
JOIN table3 t3
    ON t3.g = t2.h
WHERE t2.i = 42 AND t2.j = 24;
```

Other Things to Avoid

Don’t do this!

```
-- Avoid scalar functions in WHERE clauses
SELECT c1 from t1 WHERE function(t2) = 1;

-- Avoid dynamic SQL
DECLARE @sql NVARCHAR(MAX) = ‘SELECT c1 FROM t1’;
EXECUTE(@sql);

-- Avoid unnecessary GROUP BY and ORDER BY
SELECT DISTINCT column1, column2
FROM table1
GROUP BY column 1, column 2
ORDER BY column 1, column 2;

-- Avoid CURSORs and looping where available
```