**Indian Institute of Technology Madras Zanzibar**

School of Engineering and Science

# Z2004: Comprehensive SQL Reference

T-SQL (SQL Server) and PostgreSQL — Every Command, Every Clause

**Z2004 — Database Management Systems**

Document 4 of 6 | B.S. Data Science and Artificial Intelligence, Year 2

Even Semester: March – July 2026

**Dr. Innocent Nyalala**

Assistant Professor of Data Science and AI

Head, SAAIL Lab | IIT Madras Zanzibar

innocent@iitmz.ac.in | Office Hours: Wednesday 2:00–4:00 PM | Room 106

**Teaching Assistants**

Support hours: **Friday 2:00–4:00 PM (EAT)**

**Sri Advaita Varakavi** | zda23b010@iitmz.ac.in

**Saleh Saleh** | zda23b001@iitmz.ac.in

# Primary (Weeks 9+): T-SQL on Microsoft SQL Server 2022

PostgreSQL differences annotated as **[PG: ...]**

## Contents


1 SQL Categories and Execution Order 2

2 Data Types 2

3 Reference Schema (Used Throughout This Reference) 2

4 DDL — Data Definition Language 5

5 DML — SELECT Queries 6

6 DML — INSERT, UPDATE, DELETE, MERGE 7

7 DML — Joins 8

8 DML — Subqueries and CTEs 9

9 Set Operators (UNION, INTERSECT, EXCEPT) 10

10 Expressions, NULL Handling, and Type Conversion 11

11 Useful Built-in Functions (Quick Sheet) 11

12 Grouping Extensions and Pagination 12

13 Temporary Tables and Common Utilities 12

14 Query Plans and Performance (Basics) 13

15 JSON and Full-Text Search (Optional but Common in Modern SQL) 13

16 Window Functions 13

17 Indexes 14

18 Views 15

19 Stored Procedures and Functions 15

20 Triggers 17

21 Transactions and Concurrency 17

22 Security — DCL 18

23 Vector Database: pgvector (PostgreSQL) 19

24 Python Integration 19

## 1. SQL Categories and Execution Order

### The Five SQL Categories

| | |
|---|---|
| **DDL** | `CREATE ALTER DROP TRUNCATE` |
| **DML** | `SELECT INSERT UPDATE DELETE MERGE` |
| **DCL** | `GRANT REVOKE` |
| **TCL** | `BEGIN COMMIT ROLLBACK SAVEPOINT` |
| **DQL** | `SELECT` (sometimes treated separately) |

### SELECT Logical Execution Order

1. `FROM` + `JOIN`
2. `WHERE`
3. `GROUP BY`
4. `HAVING`
5. `SELECT`
6. `DISTINCT`
7. `ORDER BY`
8. `TOP / LIMIT / FETCH`

*You write SQL in a different order than the engine executes it.*

## 2. Data Types

### T-SQL Data Types (SQL Server)

| | |
|---|---|
| **Integer** | `INT BIGINT SMALLINT TINYINT` |
| **Decimal** | `DECIMAL(p,s) NUMERIC(p,s) MONEY` |
| **Float** | `FLOAT REAL` |
| **String** | `VARCHAR(n) NVARCHAR(n) CHAR(n)` |
| **Text** | `VARCHAR(MAX) NVARCHAR(MAX)` |
| **Date/Time** | `DATE TIME DATETIME DATETIME2` |
| **Boolean** | `BIT` (0 = false, 1 = true) |
| **Binary** | `VARBINARY(n) VARBINARY(MAX)` |
| **Other** | `UNIQUEIDENTIFIER XML ROWVERSION` |

### PostgreSQL Data Types

| | |
|---|---|
| **Integer** | `INTEGER BIGINT SMALLINT SERIAL` |
| **Decimal** | `DECIMAL(p,s) NUMERIC(p,s)` |
| **Float** | `DOUBLE PRECISION REAL` |
| **String** | `VARCHAR(n) CHAR(n) TEXT` |
| **Date/Time** | `DATE TIME TIMESTAMP TIMESTAMPTZ` |
| **Boolean** | `BOOLEAN` (TRUE/FALSE) |
| **JSON** | `JSON JSONB` |
| **Array** | `INTEGER[] TEXT[]` |
| **Vector** | `vector(n)` (via pgvector extension) |
| **UUID** | `UUID` |

## 3. Reference Schema (Used Throughout This Reference)

## Tables and Key Relationships (PK/FK)

| | |
|---|---|
| **Departments** | `DeptID` (PK) |
| **Students** | `StudentID` (PK), `DeptID` (FK → Departments.DeptID) |
| **Courses** | `CourseID` (PK) |
| **Enrollments** | `(StudentID, CourseID)` (PK), FKs to Students + Courses |
| **Orders** | `OrderID` (PK) (used for window/running-total examples) |
| **Accounts** | `AccountID` (PK) (used for transactions examples) |
| **Employees** | `EmployeeID` (PK), `ManagerID` (self-FK) (used for self-join/recursive CTE) |
| **DeptTransfers** | `StudentID` (PK, FK), `NewDeptID` (FK) (used for UPDATE...JOIN) |
| **Graduates** | `StudentID` (PK, FK) (used for INSERT...SELECT) |
| **Suspended** | `StudentID` (PK, FK) (used for DELETE...IN subquery) |
| **NewEnrollments** | staging table used for `MERGE` / upsert examples |
| **AuditLog** | `LogID` (PK) (used for trigger logging) |
| **Documents** | `DocID` (PK) (used for JSON, full-text, and pgvector examples) |

## Schema DDL (SQL Server 2022 primary; PG notes inline)

```
-- =====================
-- Core master tables
-- =====================
CREATE TABLE Departments (
    DeptID     INT           PRIMARY KEY,
    DeptName   NVARCHAR(100) NOT NULL,
    Location   NVARCHAR(100) NULL
);

CREATE TABLE Students (
    StudentID     INT            IDENTITY(1,1) PRIMARY KEY,        -- [PG]: use GENERATED ALWAYS AS IDENTITY
    FullName      NVARCHAR(100) NOT NULL,
    Email         NVARCHAR(150) NOT NULL UNIQUE,
    PhoneNumber   NVARCHAR(20)  NULL,
    DOB           DATE          NOT NULL,
    GPA           DECIMAL(3,2)  NOT NULL CHECK (GPA BETWEEN 0 AND 4),
    DeptID        INT           NULL,
    ProfileJson   NVARCHAR(MAX) NULL,                             -- [PG]: use JSONB
    CreatedAt     DATETIME2     NOT NULL DEFAULT GETDATE(),       -- [PG]: DEFAULT NOW()
    CONSTRAINT fk_students_dept
        FOREIGN KEY (DeptID) REFERENCES Departments(DeptID)
        ON DELETE SET NULL ON UPDATE CASCADE
);

CREATE TABLE Courses (
    CourseID   INT           PRIMARY KEY,
    CourseName NVARCHAR(120) NOT NULL
);

CREATE TABLE Enrollments (
    StudentID  INT NOT NULL,
    CourseID   INT NOT NULL,
    EnrolledAt DATETIME2 NOT NULL DEFAULT GETDATE(),
    CONSTRAINT pk_enrollments PRIMARY KEY (StudentID, CourseID),
    CONSTRAINT fk_enr_student FOREIGN KEY (StudentID) REFERENCES Students(StudentID) ON DELETE CASCADE,
    CONSTRAINT fk_enr_course  FOREIGN KEY (CourseID)  REFERENCES Courses(CourseID)  ON DELETE CASCADE
);
```

```
38
39  -- =====================
40  -- Tables used in specific examples
41  -- =====================
42  CREATE TABLE Orders (
43      OrderID   INT IDENTITY(1,1) PRIMARY KEY,
44      OrderDate DATE          NOT NULL,
45      Amount    DECIMAL(12,2) NOT NULL CHECK (Amount >= 0)
46  );
47
48  CREATE TABLE Accounts (
49      AccountID INT PRIMARY KEY,
50      Balance   DECIMAL(12,2) NOT NULL
51  );
52
53  CREATE TABLE Employees (
54      EmployeeID INT PRIMARY KEY,
55      Name       NVARCHAR(100) NOT NULL,
56      ManagerID  INT NULL,
57      CONSTRAINT fk_emp_manager
58          FOREIGN KEY (ManagerID) REFERENCES Employees(EmployeeID)
59  );
60
61  CREATE TABLE DeptTransfers (
62      StudentID INT PRIMARY KEY,
63      NewDeptID INT NOT NULL,
64      CONSTRAINT fk_transfer_student FOREIGN KEY (StudentID) REFERENCES
            Students(StudentID) ON DELETE CASCADE,
65      CONSTRAINT fk_transfer_dept    FOREIGN KEY (NewDeptID) REFERENCES
            Departments(DeptID)
66  );
67
68  CREATE TABLE Graduates (
69      StudentID INT PRIMARY KEY,
70      FullName  NVARCHAR(100) NOT NULL,
71      GPA       DECIMAL(3,2)  NOT NULL,
72      CONSTRAINT fk_grads_student FOREIGN KEY (StudentID) REFERENCES
            Students(StudentID)
73  );
74
75  CREATE TABLE Suspended (
76      StudentID INT PRIMARY KEY,
77      Reason    NVARCHAR(200) NULL,
78      CONSTRAINT fk_susp_student FOREIGN KEY (StudentID) REFERENCES
            Students(StudentID) ON DELETE CASCADE
79  );
80
81  CREATE TABLE NewEnrollments (
82      StudentID INT PRIMARY KEY,
83      FullName  NVARCHAR(100) NOT NULL,
84      Email     NVARCHAR(150) NOT NULL,
85      GPA       DECIMAL(3,2)  NOT NULL CHECK (GPA BETWEEN 0 AND 4),
86      DeptID    INT NULL
87  );
88
89  CREATE TABLE AuditLog (
90      LogID      INT IDENTITY(1,1) PRIMARY KEY,
91      Action     NVARCHAR(20)  NOT NULL,
92      TableName  NVARCHAR(128) NOT NULL,
93      ActionTime DATETIME2     NOT NULL DEFAULT GETDATE()
94  );
95
```

```sql
CREATE TABLE Documents (
    DocID        INT IDENTITY(1,1) PRIMARY KEY,
    Content      NVARCHAR(MAX) NOT NULL,
    MetadataJson NVARCHAR(MAX) NULL   -- [PG]: use JSONB
);

-- [PG/pgvector section]
-- CREATE EXTENSION IF NOT EXISTS vector;
-- ALTER TABLE Documents ADD COLUMN Embedding vector(384);
```

## 4. DDL — Data Definition Language

### CREATE TABLE (Pattern) *(See Reference Schema for the full course schema)*

```sql
-- Template: keys, constraints, identity/defaults
CREATE TABLE Example (
    ID         INT IDENTITY(1,1) PRIMARY KEY,      -- [PG]: GENERATED ALWAYS AS IDENTITY
    Name       NVARCHAR(100) NOT NULL,
    Email      NVARCHAR(150) NOT NULL UNIQUE,
    Score      DECIMAL(5,2)  NOT NULL CHECK (Score >= 0),
    DeptID     INT NULL,
    CreatedAt DATETIME2      NOT NULL DEFAULT GETDATE(),  -- [PG]: DEFAULT NOW()
    CONSTRAINT fk_example_dept
        FOREIGN KEY (DeptID) REFERENCES Departments(DeptID)
        ON DELETE SET NULL ON UPDATE CASCADE
);
```

### ALTER TABLE

```sql
-- Add / drop column
ALTER TABLE Students ADD    MiddleName NVARCHAR(50) NULL;
ALTER TABLE Students DROP   COLUMN MiddleName;

-- Alter column type
ALTER TABLE Students ALTER  COLUMN GPA DECIMAL(4,3);  -- T-SQL
-- [PG]: ALTER TABLE Students ALTER COLUMN GPA TYPE DECIMAL(4,3);

-- Add / drop constraint
ALTER TABLE Students ADD    CONSTRAINT chk_gpa CHECK (GPA >= 0);
ALTER TABLE Students DROP   CONSTRAINT chk_gpa;

-- Add foreign key
ALTER TABLE Students ADD    FOREIGN KEY (DeptID) REFERENCES Departments(DeptID);
```

### DROP, TRUNCATE, and RENAME

```sql
DROP TABLE Students;                -- Deletes table AND all data permanently
DROP TABLE IF EXISTS Students;      -- Safe drop (no error if not exists)
```

```
TRUNCATE TABLE Students;              -- Deletes all rows; keeps structure ; faster than DELETE
EXEC sp_rename 'Students', 'Learners';  -- T-SQL: rename table
-- [PG]: ALTER TABLE Students RENAME TO Learners;
```

### Constraints Reference

```
-- Column-level constraints
ColName INT NOT NULL,
ColName INT UNIQUE,
ColName INT PRIMARY KEY,
ColName INT DEFAULT 0,
ColName INT CHECK (ColName > 0),
ColName INT REFERENCES OtherTable(OtherCol),

-- Table-level constraints (after all columns)
CONSTRAINT pk_name    PRIMARY KEY (Col1, Col2),       -- composite PK
CONSTRAINT uq_name    UNIQUE (Col1, Col2),
CONSTRAINT fk_name    FOREIGN KEY (Col1) REFERENCES T2(C1)
                      ON DELETE CASCADE ON UPDATE SET NULL,
CONSTRAINT chk_name   CHECK (Col1 < Col2)
```

## 5. DML — SELECT Queries

### SELECT Fundamentals

```
SELECT *  FROM Students;
SELECT StudentID, FullName, GPA FROM Students;
SELECT TOP 10 * FROM Students ORDER BY GPA DESC;              -- T-SQL
-- [PG]: SELECT * FROM Students ORDER BY GPA DESC LIMIT 10;
SELECT DISTINCT DeptID FROM Students;
SELECT FullName AS Name, GPA * 4 AS ScaledGPA FROM Students;
SELECT * FROM Students WHERE GPA > 3.5 AND DeptID = 2;
SELECT * FROM Students WHERE FullName LIKE 'A%';              -- starts with A
SELECT * FROM Students WHERE GPA BETWEEN 3.0 AND 3.9;
SELECT * FROM Students WHERE DeptID IN (1, 3, 5);
SELECT * FROM Students WHERE DeptID IS NULL;
SELECT * FROM Students WHERE DeptID IS NOT NULL;
SELECT * FROM Students ORDER BY GPA DESC, FullName ASC;
```

### Aggregate Functions

```
SELECT COUNT(*)              AS TotalStudents  FROM Students;
SELECT COUNT(DeptID)         AS WithDept       FROM Students;  -- ignores NULLs
SELECT COUNT(DISTINCT DeptID) AS NumDepts      FROM Students;
SELECT SUM(GPA)              AS TotalGPA       FROM Students;
SELECT AVG(GPA)              AS AvgGPA         FROM Students;
SELECT MAX(GPA), MIN(GPA)                      FROM Students;
SELECT DeptID,
       COUNT(*)              AS NumStudents,
       AVG(GPA)              AS AvgGPA,
```

```
        MAX(GPA)              AS TopGPA
FROM    Students
GROUP   BY DeptID
HAVING  AVG(GPA) > 3.0
ORDER   BY AvgGPA DESC;
```

## 6. DML — INSERT, UPDATE, DELETE, MERGE

### </> INSERT

```
-- Single row
INSERT INTO Students (FullName, Email, DOB, GPA, DeptID)
VALUES ('Amina Hassan', 'amina@iitmz.ac.tz', '2004-03-15', 3.85, 2);

-- Multiple rows
INSERT INTO Students (FullName, Email, DOB, GPA, DeptID) VALUES
    ('John Mwanga', 'john@iitmz.ac.tz', '2003-07-22', 3.60, 1),
    ('Fatuma Ali',  'fatuma@iitmz.ac.tz', '2004-01-10', 3.92, 3);

-- Insert from SELECT
INSERT INTO Graduates (StudentID, FullName, GPA)
SELECT StudentID, FullName, GPA FROM Students WHERE GPA >= 3.5;
```

### </> UPDATE and DELETE

```
-- UPDATE with WHERE (always use WHERE!)
UPDATE Students SET GPA = 3.95 WHERE StudentID = 101;
UPDATE Students SET GPA = GPA + 0.1 WHERE DeptID = 2 AND GPA < 3.8;

-- UPDATE with JOIN (T-SQL)
-- Example: apply a transfer table to move students to a new department
UPDATE s
SET    s.DeptID = t.NewDeptID
FROM   Students s
JOIN   DeptTransfers t ON t.StudentID = s.StudentID;

-- DELETE with WHERE (always use WHERE!)
DELETE FROM Students WHERE GPA < 1.0;
DELETE FROM Students WHERE StudentID IN (SELECT StudentID FROM Suspended)
    ;

-- TRUNCATE (removes ALL rows, no rollback in most cases)
TRUNCATE TABLE Orders;
```

### </> MERGE (Upsert)

```
-- T-SQL MERGE (upsert pattern)
MERGE INTO Students AS target
USING NewEnrollments AS source ON target.StudentID = source.StudentID
WHEN MATCHED THEN
    UPDATE SET target.FullName = source.FullName,
               target.Email    = source.Email,
               target.GPA      = source.GPA,
```

```
                target.DeptID   = source.DeptID
WHEN NOT MATCHED BY TARGET THEN
    INSERT (StudentID, FullName, Email, GPA, DeptID)
    VALUES (source.StudentID, source.FullName, source.Email, source.GPA,
        source.DeptID);

-- [PG]: Use INSERT ... ON CONFLICT (equivalent upsert)
INSERT INTO Students (StudentID, FullName, Email, GPA, DeptID)
VALUES (101, 'Updated Name', 'updated@iitmz.ac.tz', 3.90, 2)
ON CONFLICT (StudentID)
DO UPDATE SET FullName = EXCLUDED.FullName,
              Email    = EXCLUDED.Email,
              GPA      = EXCLUDED.GPA,
              DeptID   = EXCLUDED.DeptID;
```

## 7. DML — Joins

### </> All JOIN Types

```
-- INNER JOIN: only matching rows
SELECT s.FullName, d.DeptName
FROM   Students s INNER JOIN Departments d ON s.DeptID = d.DeptID;

-- LEFT OUTER JOIN: all students, even those with no department
SELECT s.FullName, d.DeptName
FROM   Students s LEFT JOIN Departments d ON s.DeptID = d.DeptID;

-- RIGHT OUTER JOIN: all departments, even empty ones
SELECT s.FullName, d.DeptName
FROM   Students s RIGHT JOIN Departments d ON s.DeptID = d.DeptID;

-- FULL OUTER JOIN: all rows from both sides
SELECT s.FullName, d.DeptName
FROM   Students s FULL OUTER JOIN Departments d ON s.DeptID = d.DeptID;

-- CROSS JOIN: every student paired with every course
SELECT s.FullName, c.CourseName
FROM   Students s CROSS JOIN Courses c;

-- SELF JOIN: employee and their manager (same table)
SELECT e.Name AS Employee, m.Name AS Manager
FROM   Employees e LEFT JOIN Employees m ON e.ManagerID = m.EmployeeID;
```

### </> JOIN Patterns (Semi, Anti, APPLY/LATERAL)

```
-- SEMI JOIN: rows in A that have a match in B
SELECT s.*
FROM   Students s
WHERE  EXISTS (SELECT 1 FROM Enrollments e WHERE e.StudentID = s.
    StudentID);

-- ANTI JOIN: rows in A that do NOT have a match in B
SELECT s.*
FROM   Students s
WHERE  NOT EXISTS (SELECT 1 FROM Enrollments e WHERE e.StudentID = s.
```

```
    StudentID);

-- Join and filter: filter in ON (keeps LEFT JOIN rows) vs WHERE (can
    turn it into INNER JOIN)
SELECT s.FullName, e.CourseID
FROM   Students s
LEFT   JOIN Enrollments e
       ON e.StudentID = s.StudentID
      AND e.CourseID  = 101;    -- filter on joined table but preserve non
          -matches

-- APPLY / LATERAL (row-by-row derived table)
-- T-SQL: CROSS APPLY / OUTER APPLY
SELECT s.StudentID, x.TopCourseID
FROM   Students s
CROSS  APPLY (
    SELECT TOP 1 e.CourseID AS TopCourseID
    FROM   Enrollments e
    WHERE  e.StudentID = s.StudentID
    ORDER  BY e.EnrolledAt DESC
) x;

-- [PG]: use LATERAL instead of APPLY
-- SELECT s.studentid, x.topcourseid
-- FROM students s
-- JOIN LATERAL (
--     SELECT e.courseid AS topcourseid
--     FROM enrollments e
--     WHERE e.studentid = s.studentid
--     ORDER BY e.enrolledat DESC
--     LIMIT 1
-- ) x ON TRUE;
```

## 8. DML — Subqueries and CTEs

### </> Subquery Types

```
-- Scalar subquery (returns one value)
SELECT FullName FROM Students
WHERE GPA = (SELECT MAX(GPA) FROM Students);

-- Table subquery with IN
SELECT FullName FROM Students
WHERE DeptID IN (SELECT DeptID FROM Departments WHERE Location = '
    Zanzibar');

-- EXISTS (correlated)
SELECT d.DeptName FROM Departments d
WHERE EXISTS (
    SELECT 1 FROM Students s WHERE s.DeptID = d.DeptID AND s.GPA > 3.8
);

-- NOT EXISTS
SELECT d.DeptName FROM Departments d
WHERE NOT EXISTS (SELECT 1 FROM Students s WHERE s.DeptID = d.DeptID);

-- Correlated subquery (re-runs for each outer row)
SELECT s1.FullName, s1.GPA
```

```
FROM Students s1
WHERE s1.GPA > (SELECT AVG(s2.GPA) FROM Students s2 WHERE s2.DeptID = s1.
    DeptID);
```

### Common Table Expressions (WITH)

```
-- Simple CTE
WITH HighPerformers AS (
    SELECT StudentID, FullName, GPA
    FROM   Students
    WHERE  GPA > 3.7
)
SELECT * FROM HighPerformers ORDER BY GPA DESC;

-- Multiple CTEs
WITH DeptStats AS (
    SELECT DeptID, AVG(GPA) AS AvgGPA, COUNT(*) AS NumStudents
    FROM   Students GROUP BY DeptID
),
TopDepts AS (
    SELECT DeptID FROM DeptStats WHERE AvgGPA > 3.5
)
SELECT d.DeptName, ds.AvgGPA
FROM   Departments d
JOIN   DeptStats ds   ON d.DeptID = ds.DeptID
JOIN   TopDepts   td  ON d.DeptID = td.DeptID;

-- Recursive CTE (org chart)
WITH OrgChart AS (
    SELECT EmployeeID, Name, ManagerID, 0 AS Level
    FROM   Employees WHERE ManagerID IS NULL          -- anchor
    UNION ALL
    SELECT e.EmployeeID, e.Name, e.ManagerID, oc.Level + 1
    FROM   Employees e JOIN OrgChart oc ON e.ManagerID = oc.EmployeeID
        -- recursive
)
SELECT * FROM OrgChart ORDER BY Level, Name;
```

## 9. Set Operators (UNION, INTERSECT, EXCEPT)

### UNION / INTERSECT / EXCEPT

```
-- UNION: combine rows, remove duplicates
SELECT DeptID FROM Students
UNION
SELECT DeptID FROM Departments;

-- UNION ALL: keep duplicates (faster)
SELECT DeptID FROM Students
UNION ALL
SELECT DeptID FROM Departments;

-- INTERSECT: common rows
SELECT DeptID FROM Students
INTERSECT
```

```sql
SELECT DeptID FROM Departments;

-- EXCEPT: rows in first query not in second
SELECT DeptID FROM Students
EXCEPT
SELECT DeptID FROM Departments;

-- Notes: all SELECTs must have same number of columns and compatible types
```

## 10. Expressions, NULL Handling, and Type Conversion

### </> CASE, COALESCE, NULLIF, CAST

```sql
-- CASE (conditional)
SELECT FullName,
       CASE WHEN GPA >= 3.7 THEN 'A'
            WHEN GPA >= 3.0 THEN 'B'
            ELSE 'C/D' END AS Grade
FROM Students;

-- NULL handling
SELECT COALESCE(PhoneNumber, 'N/A') AS Phone FROM Students; -- first non-NULL
SELECT NULLIF(GPA, 0) AS GPA_or_NULL FROM Students;          -- returns NULL if GPA=0
SELECT ISNULL(PhoneNumber, 'N/A') AS Phone FROM Students;    -- T-SQL only
-- [PG]: use COALESCE(...) (no ISNULL)

-- Type conversion
SELECT CAST(GPA AS INT) AS GPATrunc FROM Students;
SELECT CONVERT(VARCHAR(10), DOB, 23) AS DOB_ISO FROM Students;  -- T-SQL
-- [PG]: SELECT TO_CHAR(DOB, 'YYYY-MM-DD') AS dob_iso FROM Students;
```

## 11. Useful Built-in Functions (Quick Sheet)

### </> String + Date/Time + Numeric

```sql
-- Strings (T-SQL / PG mostly similar)
SELECT UPPER(FullName), LOWER(FullName) FROM Students;
SELECT SUBSTRING(FullName, 1, 5) FROM Students;         -- T-SQL
-- [PG]: SELECT SUBSTRING(FullName FROM 1 FOR 5) FROM Students;
SELECT REPLACE(FullName, ' ', '-') FROM Students;
SELECT TRIM('  abc  ') AS Trimmed;
SELECT CONCAT(FullName, ' <', Email, '>') FROM Students;

-- Dates
SELECT GETDATE() AS Now;                                -- T-SQL
-- [PG]: SELECT NOW() AS now;
SELECT DATEADD(day, 7, GETDATE()) AS Plus7Days;         -- T-SQL
-- [PG]: SELECT NOW() + INTERVAL '7 days' AS plus7days;
SELECT DATEDIFF(day, DOB, GETDATE()) AS DaysAlive;      -- T-SQL
-- [PG]: SELECT (CURRENT_DATE - DOB) AS daysalive FROM Students;
```

```sql

-- Numeric
SELECT ROUND(GPA, 1), CEILING(GPA), FLOOR(GPA), ABS(GPA) FROM Students;
```

## 12. Grouping Extensions and Pagination

**</> ROLLUP / CUBE / GROUPING SETS + OFFSET/FETCH**

```sql
-- GROUPING SETS / ROLLUP / CUBE (supported in SQL Server and PostgreSQL)
SELECT DeptID, COUNT(*) AS NumStudents
FROM   Students
GROUP  BY GROUPING SETS ((DeptID), ());  -- per-dept + grand total

SELECT DeptID, COUNT(*) AS NumStudents
FROM   Students
GROUP  BY ROLLUP (DeptID);               -- per-dept + grand total

-- Pagination
SELECT *
FROM   Students
ORDER  BY StudentID
OFFSET 20 ROWS FETCH NEXT 10 ROWS ONLY;  -- T-SQL + PG

-- [PG]: also common: LIMIT 10 OFFSET 20;
```

## 13. Temporary Tables and Common Utilities

**</> Temp Tables, Variables, Sequences/Identity**

```sql
-- T-SQL temp tables / table variables
CREATE TABLE #TempTop (StudentID INT, GPA DECIMAL(3,2));
INSERT INTO #TempTop SELECT TOP 10 StudentID, GPA FROM Students ORDER BY
    GPA DESC;

DECLARE @Top TABLE (StudentID INT, GPA DECIMAL(3,2));
INSERT INTO @Top SELECT TOP 10 StudentID, GPA FROM Students ORDER BY GPA
    DESC;

-- [PG]: temporary table
-- CREATE TEMP TABLE temp_top(studentid INT, gpa NUMERIC(3,2));
-- INSERT INTO temp_top SELECT studentid, gpa FROM students ORDER BY gpa
    DESC LIMIT 10;

-- Identity / sequences
SELECT SCOPE_IDENTITY() AS LastID;                    -- T-SQL
-- [PG]: SELECT currval(pg_get_serial_sequence('students','studentid'));
```

## 14. Query Plans and Performance (Basics)

### EXPLAIN, Statistics, and Sargability

```sql
-- PostgreSQL: see plan and actual timing
EXPLAIN ANALYZE
SELECT * FROM Students WHERE DeptID = 2 AND GPA >= 3.5;

-- SQL Server: (in SSMS) include actual execution plan +
SET STATISTICS IO ON;
SET STATISTICS TIME ON;
SELECT * FROM Students WHERE DeptID = 2 AND GPA >= 3.5;

-- Sargable predicate: good (uses index)
SELECT * FROM Students WHERE DOB >= '2004-01-01';

-- Non-sargable predicate: often bad (prevents index seek)
SELECT * FROM Students WHERE YEAR(DOB) = 2004;       -- T-SQL
-- [PG]: EXTRACT(YEAR FROM dob) = 2004 has similar issue
```

## 15. JSON and Full-Text Search (Optional but Common in Modern SQL)

### JSON (T-SQL vs PG) + Full-Text

```sql
-- T-SQL JSON (stored as NVARCHAR)
SELECT JSON_VALUE(ProfileJson, '$.city') AS City FROM Students;
SELECT JSON_VALUE(MetadataJson, '$.source') AS Source FROM Documents;

-- [PG] JSONB operators (if columns are JSONB)
-- SELECT profilejson ->> 'city' AS city FROM students;
-- SELECT * FROM students WHERE profilejson @> '{"city":"Zanzibar"}';
-- SELECT metadatajson ->> 'source' AS source FROM documents;

-- Full-text search
-- T-SQL (requires full-text index): CONTAINS/CONTAINSTABLE
-- SELECT * FROM Documents WHERE CONTAINS(Content, '"database"');

-- [PG] full-text search
-- SELECT * FROM documents WHERE to_tsvector('english', content) @@
    plainto_tsquery('english', 'database');
```

## 16. Window Functions

### Window Function Syntax and All Types

```sql
-- Syntax template
function_name() OVER (
    [PARTITION BY col1, col2]    -- divide into groups (optional)
    [ORDER BY col3 DESC]         -- ordering within each partition
    [ROWS|RANGE BETWEEN ...]     -- frame specification (optional)
)

```

```sql
-- Ranking functions
SELECT FullName, GPA, DeptID,
    ROW_NUMBER()   OVER (PARTITION BY DeptID ORDER BY GPA DESC) AS RowNum,
    RANK()         OVER (PARTITION BY DeptID ORDER BY GPA DESC) AS Rnk,
    DENSE_RANK()   OVER (PARTITION BY DeptID ORDER BY GPA DESC) AS DRnk,
    NTILE(4)       OVER (ORDER BY GPA DESC)                     AS Quartile,
    PERCENT_RANK() OVER (ORDER BY GPA)                          AS PctRank,
    CUME_DIST()    OVER (ORDER BY GPA)                          AS CumDist
FROM Students;

-- Offset functions
SELECT OrderDate, Amount,
    LAG(Amount, 1, 0)  OVER (ORDER BY OrderDate) AS PrevAmount,
    LEAD(Amount, 1, 0) OVER (ORDER BY OrderDate) AS NextAmount,
    FIRST_VALUE(Amount) OVER (ORDER BY OrderDate
        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS FirstInWindow,
    LAST_VALUE(Amount)  OVER (ORDER BY OrderDate
        ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING) AS LastInWindow
FROM Orders;

-- Aggregate window functions (per-row aggregates without collapsing rows)
SELECT DeptID, FullName, GPA,
    AVG(GPA)  OVER (PARTITION BY DeptID) AS DeptAvgGPA,
    COUNT(*)  OVER (PARTITION BY DeptID) AS DeptSize
FROM Students;

-- Running total (typical finance/orders example)
SELECT OrderDate, Amount,
    SUM(Amount) OVER (
        ORDER BY OrderDate
        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
    ) AS RunningTotal
FROM Orders;
```

## 17. Indexes

**</> Index Creation and Management**

```sql
-- B-Tree index (default; best for range and equality queries)
CREATE INDEX idx_students_gpa ON Students (GPA);
CREATE INDEX idx_students_dept_gpa ON Students (DeptID, GPA);  -- composite

-- Unique index
CREATE UNIQUE INDEX idx_students_email ON Students (Email);

-- Clustered index (T-SQL; one per table; determines physical order)
CREATE CLUSTERED INDEX idx_students_pk ON Students (StudentID);

-- Non-clustered with INCLUDE (covering index)
CREATE NONCLUSTERED INDEX idx_cover ON Students (DeptID)
INCLUDE (FullName, GPA);      -- extra columns in leaf level; avoids
```

```
    table lookup

-- [PG]: All T-SQL index syntax is nearly identical in PostgreSQL
-- [PG]: CREATE INDEX CONCURRENTLY is available (non-locking)
CREATE INDEX CONCURRENTLY idx_students_gpa ON Students (GPA);  -- PG only

-- Hash index (equality queries only)
CREATE INDEX idx_hash ON Students USING HASH (Email);  -- PG syntax
-- T-SQL: hash indexes only available on memory-optimised tables

-- Drop index
DROP INDEX idx_students_gpa ON Students;    -- T-SQL
DROP INDEX idx_students_gpa;                -- PG (no table name needed)

-- View index usage
SELECT * FROM sys.indexes WHERE object_id = OBJECT_ID('Students'); -- T-
    SQL
```

## 18. Views

**</> Views**

```
-- Create a view
CREATE VIEW vw_HighPerformers AS
SELECT s.FullName, s.GPA, d.DeptName
FROM   Students s JOIN Departments d ON s.DeptID = d.DeptID
WHERE  s.GPA > 3.5;

-- Query a view exactly like a table
SELECT * FROM vw_HighPerformers ORDER BY GPA DESC;

-- Updatable view (simple views can accept INSERT/UPDATE/DELETE)
UPDATE vw_HighPerformers SET GPA = 3.9 WHERE FullName = 'Amina Hassan';

-- Alter a view (T-SQL)
ALTER VIEW vw_HighPerformers AS
SELECT s.FullName, s.GPA, s.Email, d.DeptName
FROM   Students s JOIN Departments d ON s.DeptID = d.DeptID
WHERE  s.GPA > 3.0;

-- Drop view
DROP VIEW IF EXISTS vw_HighPerformers;
```

## 19. Stored Procedures and Functions

**</> Stored Procedures (T-SQL)**

```
CREATE PROCEDURE GetStudentsByDept
    @DeptID INT,
    @MinGPA DECIMAL(3,2) = 0.0     -- default parameter
AS
BEGIN
    SET NOCOUNT ON;
```

```
    SELECT StudentID, FullName, GPA
    FROM   Students
    WHERE  DeptID = @DeptID AND GPA >= @MinGPA
    ORDER  BY GPA DESC;
END;
GO

-- Execute
EXEC GetStudentsByDept @DeptID = 2, @MinGPA = 3.5;

-- Drop procedure
DROP PROCEDURE IF EXISTS GetStudentsByDept;
```

**Functions (T-SQL)**

```
-- Scalar function
CREATE FUNCTION dbo.GetLetterGrade (@GPA DECIMAL(3,2))
RETURNS NVARCHAR(2)
AS BEGIN
    RETURN CASE
        WHEN @GPA >= 3.7 THEN 'A'
        WHEN @GPA >= 3.0 THEN 'B'
        WHEN @GPA >= 2.0 THEN 'C'
        ELSE 'D' END;
END;
GO

SELECT FullName, GPA, dbo.GetLetterGrade(GPA) AS Grade FROM Students;

-- Table-valued function
CREATE FUNCTION dbo.GetDeptStudents (@DeptID INT)
RETURNS TABLE AS RETURN
    SELECT StudentID, FullName, GPA FROM Students WHERE DeptID = @DeptID;
```

**PostgreSQL Functions (PL/pgSQL)**

```
CREATE OR REPLACE FUNCTION get_dept_avg(p_dept_id INT)
RETURNS DECIMAL AS $$
DECLARE
    v_avg DECIMAL;
BEGIN
    SELECT AVG(GPA) INTO v_avg FROM Students WHERE DeptID = p_dept_id;
    RETURN v_avg;
END;
$$ LANGUAGE plpgsql;

SELECT get_dept_avg(2);
```

## 20. Triggers

**Triggers (T-SQL)**

```
-- AFTER INSERT trigger: log new student enrollment
CREATE TRIGGER trg_AfterStudentInsert
ON Students
AFTER INSERT
AS BEGIN
    SET NOCOUNT ON;
    INSERT INTO AuditLog (Action, TableName, ActionTime)
    SELECT 'INSERT', 'Students', GETDATE()
    FROM   inserted;   -- virtual table with newly inserted rows
END;
GO

-- AFTER UPDATE trigger: prevent GPA from being lowered more than 1 point
CREATE TRIGGER trg_PreventGPADrop
ON Students
AFTER UPDATE
AS BEGIN
    IF EXISTS (
        SELECT 1 FROM inserted i JOIN deleted d ON i.StudentID = d.
            StudentID
        WHERE  d.GPA - i.GPA > 1.0
    )
    BEGIN
        ROLLBACK;
        RAISERROR('GPA cannot be lowered by more than 1.0 point.', 16, 1)
            ;
    END;
END;
GO

-- [PG]: Triggers use a two-step approach (function + trigger)
CREATE OR REPLACE FUNCTION check_gpa_drop() RETURNS TRIGGER AS $$
BEGIN
    IF OLD.GPA - NEW.GPA > 1.0 THEN
        RAISE EXCEPTION 'GPA cannot drop by more than 1.0';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_gpa_check BEFORE UPDATE ON Students
FOR EACH ROW EXECUTE FUNCTION check_gpa_drop();
```

## 21. Transactions and Concurrency

**Transactions (T-SQL)**

```
-- Basic transaction
BEGIN TRANSACTION;
    UPDATE Accounts SET Balance = Balance - 500 WHERE AccountID = 1;
    UPDATE Accounts SET Balance = Balance + 500 WHERE AccountID = 2;
    IF @@ERROR <> 0
        ROLLBACK;
```

```
    ELSE
        COMMIT;

-- TRY...CATCH transaction (recommended)
BEGIN TRY
    BEGIN TRANSACTION;
        UPDATE Accounts SET Balance = Balance - 500 WHERE AccountID = 1;
        UPDATE Accounts SET Balance = Balance + 500 WHERE AccountID = 2;
    COMMIT TRANSACTION;
END TRY
BEGIN CATCH
    ROLLBACK TRANSACTION;
    THROW;
END CATCH;

-- SAVEPOINT
SAVEPOINT sp1;          -- PG syntax
ROLLBACK TO sp1;        -- PG syntax
SAVE TRANSACTION sp1;   -- T-SQL
ROLLBACK TRANSACTION sp1; -- T-SQL

-- Isolation levels (SET before BEGIN TRANSACTION)
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;   -- dirty reads allowed
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;     -- default in most DBs
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;    -- no non-repeatable
    reads
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;       -- strictest; no
    phantoms
SET TRANSACTION ISOLATION LEVEL SNAPSHOT;           -- T-SQL only (MVCC)
```

## 22. Security — DCL

**</> GRANT and REVOKE**

```
-- Grant specific privileges
GRANT SELECT, INSERT ON Students TO ReadWriteRole;
GRANT EXECUTE ON GetStudentsByDept TO AppUser;
GRANT ALL PRIVILEGES ON Students TO AdminUser;        -- PG syntax
GRANT ALL ON Students TO AdminUser;                   -- T-SQL syntax

-- Grant with propagation
GRANT SELECT ON Students TO ReaderRole WITH GRANT OPTION;

-- Revoke privileges
REVOKE INSERT ON Students FROM ReadWriteRole;
REVOKE ALL ON Students FROM AppUser;

-- T-SQL: Create a login, user, and role
CREATE LOGIN AppLogin WITH PASSWORD = 'Str0ng!Pass';
CREATE USER  AppUser  FOR LOGIN AppLogin;
CREATE ROLE  ReaderRole;
ALTER ROLE   ReaderRole ADD MEMBER AppUser;
GRANT SELECT ON SCHEMA::dbo TO ReaderRole;
```

## 23. Vector Database: pgvector (PostgreSQL)

**pgvector Setup and Queries**

```sql
-- Install extension (once per database)
CREATE EXTENSION IF NOT EXISTS vector;

-- Add a vector column to the existing Documents table from the Reference Schema
-- (Documents.Content stores text; we add Documents.Embedding for embeddings)
ALTER TABLE Documents
ADD COLUMN Embedding vector(384);  -- 384-dim for all-MiniLM-L6-v2 model

-- Insert a vector (from Python: pass as list)
INSERT INTO Documents (Content, Embedding)
VALUES ('Introduction to databases', '[0.12, -0.03, 0.45, ...]');

-- Cosine similarity search (most similar documents)
SELECT DocID, Content,
       1 - (Embedding <=> '[0.10, -0.05, 0.48, ...]') AS CosineSim
FROM   Documents
ORDER  BY Embedding <=> '[0.10, -0.05, 0.48, ...]'
LIMIT  5;

-- Operators: <=> cosine distance | <-> L2 (Euclidean) | <#> dot product

-- HNSW index for fast approximate nearest-neighbour search
CREATE INDEX idx_hnsw ON Documents
USING hnsw (Embedding vector_cosine_ops)
WITH (m = 16, ef_construction = 64);

-- IVFFlat index (alternative)
CREATE INDEX idx_ivfflat ON Documents
USING ivfflat (Embedding vector_cosine_ops) WITH (lists = 100);
```

## 24. Python Integration

**Python + psycopg2 (PostgreSQL)**

```python
import psycopg2

conn = psycopg2.connect(
    host="localhost", database="z2004_db",
    user="postgres", password="your_password"
)
cur = conn.cursor()

# Parameterised query (ALWAYS use parameterised; never f-strings for SQL)
cur.execute("SELECT * FROM Students WHERE DeptID = %s AND GPA > %s", (2, 3.5))
rows = cur.fetchall()
for row in rows:
    print(row)

# Insert with parameters
```

```
cur.execute("""
    INSERT INTO Students (FullName, Email, GPA, DeptID)
    VALUES (%s, %s, %s, %s)
""", ("Amina Hassan", "amina@iitmz.ac.tz", 3.85, 2))
conn.commit()

cur.close()
conn.close()
```

### Python + SQLAlchemy ORM

```
from sqlalchemy import create_engine, Column, Integer, String, Float
from sqlalchemy.orm import declarative_base, Session

engine = create_engine("postgresql://postgres:pass@localhost/z2004_db")
Base = declarative_base()

class Student(Base):
    __tablename__ = "students"
    StudentID = Column(Integer, primary_key=True)
    FullName  = Column(String(100), nullable=False)
    GPA       = Column(Float)

Base.metadata.create_all(engine)

with Session(engine) as session:
    new_s = Student(FullName="Fatuma Ali", GPA=3.92)
    session.add(new_s)
    session.commit()

    high_gpa = session.query(Student).filter(Student.GPA > 3.7).all()
    for s in high_gpa:
        print(s.FullName, s.GPA)
```

### Python + pgvector (RAG Pipeline Snippet)

```
from sentence_transformers import SentenceTransformer
import psycopg2

model = SentenceTransformer("all-MiniLM-L6-v2")  # 384 dimensions
conn  = psycopg2.connect("postgresql://postgres:pass@localhost/z2004_db")
cur   = conn.cursor()

# Store embedding
text  = "IIT Madras Zanzibar offers B.S. in Data Science."
vec   = model.encode(text).tolist()
cur.execute("INSERT INTO Documents (Content, Embedding) VALUES (%s, %s)",
            (text, vec))
conn.commit()

# Query: find 3 most similar documents to a question
question = "What programmes does IIT Madras Zanzibar offer?"
q_vec    = model.encode(question).tolist()
cur.execute("""
    SELECT Content, 1 - (Embedding <=> %s) AS Similarity
    FROM   Documents
    ORDER  BY Embedding <=> %s
    LIMIT  3
```

```
""", (q_vec, q_vec))
results = cur.fetchall()
for content, sim in results:
    print(f"[{sim:.3f}] {content}")
```