# TSQL ICA - kleeuwkent

## SQL Server Practitioner Details:

a) Introduction to the SQL Practitioner:

- The SQL Practitioner is a professional who has mastered the fundamentals of Structured Query Language (SQL), a powerful language used for managing data in relational database management systems.
- The SQL Practitioner is responsible for developing and maintaining databases, writing queries, and troubleshooting any issues related to the database.
- They must have an understanding of database design principles and be able to use their knowledge of SQL to create efficient, secure, and reliable databases.

b) Why you should learn SQL:

- Learning SQL is a great way to gain a better understanding of how databases work and how to use them effectively.
- With the right knowledge, you can create powerful applications that can help you manage data more efficiently and store it securely.
- Additionally, having an understanding of SQL will make you a more employable candidate in the tech industry.

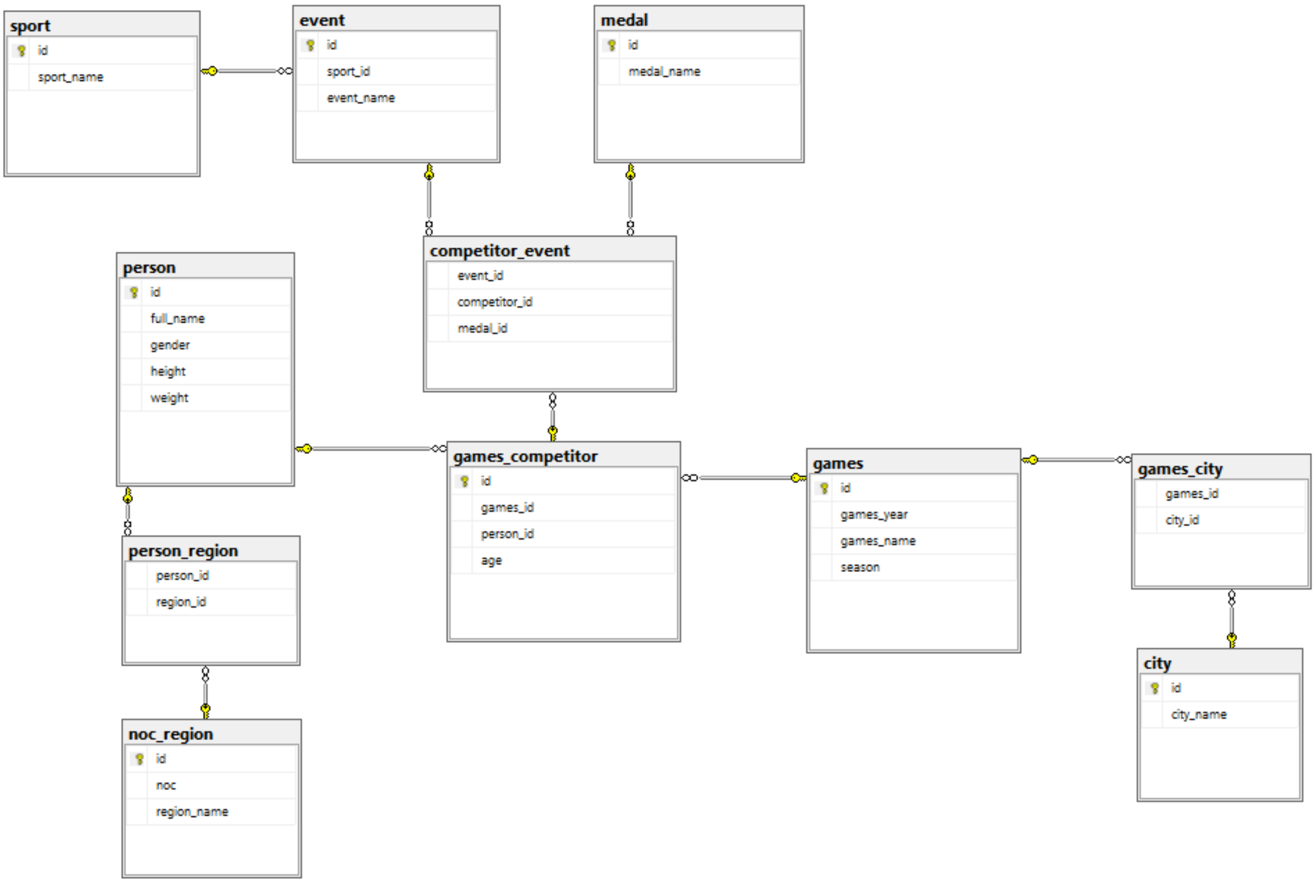
---

## SQL Server Database Overview:

a) SQL Server Database Diagrams:

- Using the sample database "Olympics" which contains the last 120 years of Olympic winners
- Data includes information on athletes who have won medals in the Summer and Winter Olympics since 1896
- Data includes events competed in, countries represented, and medals won
- Database can be used to compare medal-winning performances across different countries, sports, and disciplines

### ERD Diagram:



## Detailed Table Data:

```
dbo.city:
id (PKT int, not null)
city_name (varchar(200, null)

dbo.competitor_event:
event_id (FK int, null)
competitor_id (FK int, null)
medal_id (FK int, null)

dbo.event:
id (PK. int, not null)
sport_id (FK, int, null)
event_name (varcharen, null)

dbo.games:
TO id (PK. int. not null)
games_year (int. null)
games_name (varchar(100), null)
season (varchar(100), null)

dbo.games_city:
games_id (FKI int, null)
city_id (FK, int, null)

dbo.games_competitor:
id (PKT int, not null)
```

```
games_id (FK, int, null)
person_id (FK, int, null)
age (int, null)

dbo.medal:
id (PK, int, not null)
medal_name (varchar(50), null)

dbo.noc_region:
id (PK int, not null)
noc (varchar(5), null)
region_name (varchar(200), null)

dbo.person:
id (PK int, not null)
full_name (varchar(500), null)
gender (varchar(10), null)
height (int, null)
weight (int, null)

dbo.person_region:
person_id (FK int, null)
region_id (FK int null)

dbo.sport:
id (PK int, not null)
sport_name (varchar(200), null)
```

---

## TSQL Part 1: SQL Server Coding Basics:

## TSQL03 to TSQL08: SQL Server Basics:

### Module 3: Writing SELECT Queries with single Table:

- Writing SELECT queries with single table is important for engineering jobs because it allows you to quickly and accurately retrieve data from a single source.
- This can be especially useful when needing to analyze complex data or when needing to make decisions based on the data retrieved.
- It also helps to ensure that the data is accurate and up to date, which is essential in engineering jobs.
- Additionally, writing SELECT queries with single table helps to increase efficiency as it eliminates the need to search through multiple sources of data.

**Demo A1: Writing Simple SELECT Query:**

---

Query:

```sql
SELECT city_name
FROM dbo.city;
```

Output:

```
(42 rows affected)

Completion time: 2023-10-12T10:50:14.3288215+01:00
```

Return Data:

| City Name |
| --- |
| Barcelona |
| London |
| Antwerpen |
| Paris |
| Calgary |

...

---

Query:

```sql
SELECT event_name, sport_id
FROM dbo.event;
```

Output:

```
(757 rows affected)

Completion time: 2023-10-12T10:56:56.3198347+01:00
```

Return Data:

| Event Name | Sport ID |
| --- | --- |
| Basketball | 9 |
| Judo Men's Extra-Lightweight | 33 |
| Football | 25 |
| Tug-Of-War | 62 |
| Speed Skating Women's 500 metres | 54 |

| Event Name | Sport ID |
|---|---|
| ... | ... |

---

Query:

```
SELECT games_year, games_name, season
FROM dbo.games;
```

Output:

```
(51 rows affected)

Completion time: 2023-10-12T10:58:49.2380500+01:00
```

Return Data:

| games_year | games_name | season |
|---|---|---|
| 1992 | 1992 Summer | Summer |
| 2012 | 2012 Summer | Summer |
| 1920 | 1920 Summer | Summer |
| 1900 | 1900 Summer | Summer |
| 1988 | 1988 Winter | Winter |
| ... | ... | ... |

---

**Demo A2: Eliminating Duplicates with DISTINCT:**

---

Query Without DISTINCT:

```
SELECT season
FROM dbo.games;
```

Output:

```
(51 rows affected)

Completion time: 2023-10-19T09:21:56.9049219+01:00
```

Return Data:

| Season |
|---|
| Summer |
| Summer |
| Summer |
| Summer |
| Winter |

---

Query With DISTINCT:

```
SELECT DISTINCT season
FROM dbo.games;
```

Output:

```
(2 rows affected)

Completion time: 2023-10-12T11:15:02.0319216+01:00
```

Return Data:

| Season |
|---|
| Summer |
| Winter |

---

Query Without DISTINCT:

```
SELECT sport_id
FROM dbo.event;
```

Output:

```
(66 rows affected)

Completion time: 2023-10-12T11:18:53.8945653+01:00
```

Return Data:

| sport_id |
|---|
| 54 |
| 54 |
| 18 |
| 18 |
| 18 |
| ... |

---

Query With DISTINCT:

```
SELECT DISTINCT sport_id
FROM dbo.event;
```

Output:

```
(66 rows affected)

Completion time: 2023-10-12T11:18:53.8945653+01:00
```

Return Data:

| sport_id |
|---|
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |
| ... |

---

#### Demo A3: Using Column and Table Aliases Lesson

---

Query:

```
SELECT c.city_name, gc.games_id, p.full_name, m.medal_name
FROM dbo.city AS c
INNER JOIN dbo.games_city AS gc ON c.id = gc.city_id
INNER JOIN dbo.games_competitor AS gcp ON gc.games_id = gcp.games_id
INNER JOIN dbo.person AS p ON gcp.person_id = p.id
INNER JOIN dbo.competitor_event AS ce ON gcp.person_id = ce.competitor_id
INNER JOIN dbo.medal AS m ON ce.medal_id = m.id;
```

Output:

```
(265018 rows affected)

Completion time: 2023-10-19T09:30:38.5251880+01:00
```

Return Data:

| City Name | Games ID | Full Name | Medal Name |
|---|---|---|---|
| Barcelona | 1 | A Dijiang | NA |
| London | 2 | A Lamusi | NA |
| Antwerpen | 3 | Gunnar Nielsen Aaby | NA |
| Paris | 4 | Edgar Lindenau Aabye | Gold |
| Calgary | 5 | Christine Jacoba Aaftink | NA |
| Albertville | 6 | Christine Jacoba Aaftink | NA |
| Lillehammer | 7 | Christine Jacoba Aaftink | NA |
| ... | ... | ... | ... |

---

### Demo A4: Writing Simple CASE Expressions

---

Query:

```
SELECT
    CASE
        WHEN medal_id = 1 THEN 'Gold'
        WHEN medal_id = 2 THEN 'Silver'
        WHEN medal_id = 3 THEN 'Bronze'
        ELSE 'No Medal'
    END AS Medal
FROM dbo.competitor_event;
```

Output:

```
(260971 rows affected)

Completion time: 2023-10-19T09:35:30.7045404+01:00
```

Return Data:

| Medal |
|---|
| No Medal |
| No Medal |
| No Medal |
| Gold |
| No Medal |

...

---

## Module 4: Joining and Querying Multiple Tables

### Demo B1: How to provide data from 2 related tables with a Join

---

Query:

```
SELECT g.games_year, g.games_name, g.season, ce.event_id
FROM dbo.games g
JOIN dbo.competitor_event ce ON g.id = ce.event_id
```

Output:

```
(59920 rows affected)

Completion time: 2023-10-19T09:55:03.0229889+01:00
```

Return Data:

| games_year | games_name | season | event_id |
|---|---|---|---|
| 1992 | 1992 Summer | Summer | 1 |
| 2012 | 2012 Summer | Summer | 2 |
| 1920 | 1920 Summer | Summer | 3 |
| 1900 | 1900 Summer | Summer | 4 |

| games_year | games_name | season | event_id |
|---|---|---|---|
| 1988 | 1988 Winter | Winter | 5 |
| ... | ... | ... | ... |

---

#### Demo B2: How to Query with Inner Joins

---

Query:

```
SELECT c.city_name, g.games_year
FROM dbo.city c
INNER JOIN dbo.games_city gc ON c.id = gc.city_id
INNER JOIN dbo.games g ON gc.games_id = g.id
```

Output:

```
(52 rows affected)

Completion time: 2023-10-19T09:40:29.7298306+01:00
```

Return Data:

| City Name | Games Year |
|---|---|
| Barcelona | 1992 |
| London | 2012 |
| Antwerpen | 1920 |
| Paris | 1900 |
| Calgary | 1988 |
| ... | ... |

---

#### Demo B3: How to Query with Outer Joins

---

Query:

```
SELECT gcp.age, p.full_name
FROM dbo.games_competitor AS gcp
LEFT OUTER JOIN dbo.person AS p ON p.id = gcp.person_id
```

Output:

```
(180252 rows affected)

Completion time: 2023-10-19T09:59:59.4151085+01:00
```

Return Data:

| age | full_name |
|---|---|
| 24 | A Dijiang |
| 23 | A Lamusi |
| 24 | Gunnar Nielsen Aaby |
| 34 | Edgar Lindenau Aabye |
| 21 | Christine Jacoba Aaftink |
| ... | ... |

---

#### Demo B4: How Query with Cross Joins and Self Joins

---

Query:

```sql
SELECT c.id, c.city_name, g.id, g.games_year, g.games_name, g.season
FROM dbo.city c
CROSS JOIN dbo.games g;
```

Output:

```
(2142 rows affected)

Completion time: 2023-10-19T10:08:22.9105966+01:00
```

Return Data:

| id | games_year | games_name | season |
|---|---|---|---|
| 51 | 1896 | 1896 Summer | Summer |
| 4 | 1900 | 1900 Summer | Summer |
| 44 | 1904 | 1904 Summer | Summer |
| 45 | 1906 | 1906 Summer | Summer |

| id | games_year | games_name | season |
|---|---|---|---|
| 47 | 1908 | 1908 Summer | Summer |
| ... | .... | .... | .... |

---

## Module 5: Sorting and Filtering Data

### Demo C1: How to Sort Data

---

Query:

```
SELECT p.full_name ,p.gender ,p.height ,p.weight
FROM dbo.person as p
order by weight desc
```

Output:

```
(128854 rows affected)

Completion time: 2023-10-26T10:28:33.0633471+01:00
```

Return Data:

| full_name | gender | height | weight |
|---|---|---|---|
| Ricardo Blas, Jr. | M | 183 | 214 |
| Aytami Ruano Vega | M | 200 | 198 |
| Marek Galiski | M | 200 | 190 |
| Christopher J. "Chris"" Taylor" | M | 196 | 182 |
| Valentyn Rusliakov | M | 187 | 180 |
| ... | ... | ... | ... |

---

### Demo C2: How to Filter Data with Predicates

---

Query:

```
SELECT p.full_name, p.gender, p.height, p.weight
FROM dbo.person AS p
WHERE p.gender = 'M' AND p.height > 180 AND p.weight < 80;
```

Output:

```
(10515 rows affected)

Completion time: 2023-10-26T10:25:47.4424267+01:00
```

Return Data:

| full_name | gender | height | weight |
|---|---|---|---|
| Per Knut Aaland | M | 188 | 75 |
| John Aalberg | M | 183 | 72 |
| Jorma Ilmari Aalto | M | 182 | 77 |
| Pepijn Aardewijn | M | 189 | 72 |
| Mika Lauri Aarnikka | M | 187 | 76 |
| ... | ... | ... | ... |

#### Demo C3: How to Filter Data with TOP and OFFSET-FETCH

Query:

```
SELECT TOP(10) p.full_name, p.gender, p.height, p.weight
FROM dbo.person AS p
ORDER BY p.full_name;
```

```
SELECT p.full_name, p.gender, p.height, p.weight
FROM dbo.person AS p
ORDER BY p.full_name
OFFSET 10 ROWS FETCH NEXT 10 ROWS ONLY;
```

Output:

```
(10 rows affected)

Completion time: 2023-10-26T10:32:41.8330498+01:00
```

```
(10 rows affected)

Completion time: 2023-10-26T10:31:44.3769641+01:00
```

Return Data:

| full_name | gender | height | weight |
|---|---|---|---|
| A Dijiang | M | 180 | 80 |
| A Lamusi | M | 170 | 60 |
| A. Aanantha Sambu Mayavo | M | 0 | 0 |
| A. J. Tyronne Benildus "Benny"" Fernando" | M | 179 | 70 |
| A. Joshua "Josh"" West" | M | 207 | 105 |
| A. W. Nancy "Nan"" Rae" | F | 156 | 53 |
| Aa Bela Joaquim | F | 0 | 0 |
| Aadam Ismaeel Khamis | M | 172 | 67 |
| Aadjijatmiko Finarsih H. | F | 0 | 0 |
| Aadolf Fredrik Svanstrm | M | 179 | 70 |
| ... | ... | ... | ... |

| full_name | gender | height | weight |
|---|---|---|---|
| Aafke Hament | F | 181 | 64 |
| Aage Albert Leidersdorff | M | 168 | 0 |
| Aage Avaldorff Meyer | M | 0 | 0 |
| Aage Berntsen | M | 0 | 0 |
| Aage Birch | M | 172 | 70 |
| Aage Brge Poulsen | M | 185 | 68 |
| Aage Carl Christian Lassen | M | 181 | 62 |
| Aage Emil Brix | M | 0 | 0 |
| Aage Emil Kirkegaard | M | 0 | 0 |
| Aage Ernst Larsen | M | 0 | 0 |
| ... | ... | ... | ... |

---

### Demo C4: How to work with Unknown Values

---

Query:

```
SELECT *
FROM dbo.person
WHERE height IS NULL OR weight IS NULL
```

Output:

```
(0 rows affected)

Completion time: 2024-01-04T01:20:14.4590986+00:00
```

# Return Data: No Matching data

## Module 6: Working with Data Types

### Demo D1: Working with Data Type examples

---

Query:

```
-- Declare a variable of type INT
DECLARE @myInt INT;

-- Assign a value to the variable
SET @myInt = 5;

-- Print the value of the variable
PRINT @myInt;

-- Declare a variable of type VARCHAR
DECLARE @myString VARCHAR(50);

-- Assign a value to the variable
SET @myString = 'Hello World!';

-- Print the value of the variable
PRINT @myString
```

Output:

```
5
Hello World!

Completion time: 2024-01-04T01:42:53.2438476+00:00
```

---

#### Demo D2: Working with Character Data

---

Query:

```

```

Output:

```

```

Return Data:

---

#### Demo D3: Working with Date and Time Data

---

Query:

```
-- Get the current date and time
SELECT GETDATE() AS 'Current Date and Time';

-- Get the current date
SELECT CAST(GETDATE() AS DATE) AS 'Current Date';

-- Get the current time
SELECT CAST(GETDATE() AS TIME) AS 'Current Time';

-- Add 5 days to the current date
SELECT DATEADD(DAY, 5, GETDATE()) AS 'Date + 5 Days';

-- Subtract 5 days from the current date
SELECT DATEADD(DAY, -5, GETDATE()) AS 'Date - 5 Days';

-- Add 1 hour to the current time
SELECT DATEADD(HOUR, 1, GETDATE()) AS 'Time + 1 Hour';

-- Subtract 1 hour from the current time
SELECT DATEADD(HOUR, -1, GETDATE()) AS 'Time - 1 Hour
```

---

## Module 7: Using DML to Modify Data

#### Demo E1: Adding Data to Tables

---

Query:

```sql
INSERT INTO dbo.competitor_event (event_id, competitor_id, medal_id)
VALUES
    (1, 101, 201),
    (2, 102, 202)
```

---

#### Demo E2: Modifying and Removing Data

---

Query:

```sql
-- Update medal_id for a specific competitor and event
UPDATE dbo.competitor_event
SET medal_id = 203
WHERE competitor_id = 101 AND event_id = 1;
```

---

#### Demo E3: Generating Automatic Column Values

---

Query:

```sql
-- Assuming id is an identity column
INSERT INTO dbo.event (sport_id, event_name)
VALUES (301, 'NewEventName');
```

---

### Module 8: Using Built-In Functions

#### Demo F1: Writing Queries with Built-In Functions

---

Query:

```sql
SELECT g.games_name, AVG(gc.age) AS average_age
FROM dbo.games_competitor gc
JOIN dbo.games g ON gc.games_id = g.id
GROUP BY g.games_name;
```

---

#### Demo F2: Using Conversion Functions

---

Query:

```sql
SELECT full_name, age, CAST(age AS DECIMAL(5, 2)) AS age_decimal
FROM dbo.games_competitor;
```

```sql
SELECT games_name, FORMAT(games_year, 'yyyy') AS formatted_games_year
FROM dbo.games;
```

---

#### Demo F3: Using Logical Functions

---

Query:

```sql
SELECT full_name, age,
       CASE
           WHEN age < 18 THEN 'Junior'
           WHEN age BETWEEN 18 AND 30 THEN 'Young Adult'
           ELSE 'Senior'
       END AS age_category
FROM dbo.games_competitor;
```

---

#### Demo F4: Using Functions to Work with NULL

---

Query:

```sql
SELECT full_name, NULLIF(weight, 0) AS actual_weight
FROM dbo.person;
```

---

# TSQL Part 2: SQL Server Coding Functions and Features

## TSQL09: Group and Aggregating Data

#### ICA Demo 1: Using Aggregate Functions

---

Query:

```sql
SELECT AVG(height) AS average_height
FROM dbo.person;
```

#### ICA Demo 2: Using the GROUP BY Clause

Query:

```sql
SELECT c.city_name, COUNT(gc.id) AS competitor_count
FROM dbo.city c
JOIN dbo.games_competitor gc ON c.id = gc.city_id
GROUP BY c.city_name;
```

#### ICA Demo 3: Filtering Groups with HAVING

Query:

```sql
SELECT c.city_name, COUNT(gc.id) AS competitor_count
FROM dbo.city c
JOIN dbo.games_competitor gc ON c.id = gc.city_id
GROUP BY c.city_name
HAVING COUNT(gc.id) > 5;
```

### TSQL10: Using Subqueries

#### ICA Demo 1: Writing Self-Contained Subqueries

Query:

```sql
SELECT full_name, age
FROM dbo.games_competitor gc
WHERE age > (SELECT AVG(age) FROM dbo.games_competitor);
```

#### ICA Demo 2: Writing Correlated Subqueries

Query:

```sql
SELECT full_name, age
FROM dbo.games_competitor gc
WHERE age > (SELECT AVG(age) FROM dbo.games_competitor WHERE city_id =
gc.city_id);
```

---

#### ICA Demo 3: Using the EXISTS Predicate with Subqueries

---

Query:

```sql
SELECT city_name
FROM dbo.city c
WHERE EXISTS (
    SELECT 1
    FROM dbo.games_competitor
    WHERE city_id = c.id
);
```

---

### TSQL11: Using Table Expressions

#### ICA Demo 1: Using Views

---

Query:

```sql
-- Creating a view
CREATE VIEW CompetitorCityView AS
SELECT pc.full_name, pc.age, c.city_name
FROM dbo.games_competitor pc
JOIN dbo.city c ON pc.city_id = c.id;

-- Querying the view
SELECT *
FROM CompetitorCityView;
```

---

#### ICA Demo 2: Using Inline TVFs

---

Query:

```sql
-- Creating an inline TVF
CREATE FUNCTION GetCompetitorsInCity (@cityId INT)
```

```sql
RETURNS TABLE
AS
RETURN
(
    SELECT pc.full_name, pc.age
    FROM dbo.games_competitor pc
    WHERE pc.city_id = @cityId
);

-- Using the inline TVF
SELECT *
FROM dbo.GetCompetitorsInCity(1); -- Replace 1 with the desired city_id
```

---

#### ICA Demo 3: Using Derived Tables

---

Query:

```sql
-- Using a derived table
SELECT dt.full_name, dt.age, dt.city_name
FROM (
    SELECT pc.full_name, pc.age, c.city_name
    FROM dbo.games_competitor pc
    JOIN dbo.city c ON pc.city_id = c.id
) AS dt;
```

---

#### ICADemo 4: Using CTEs

---

Query:

```sql
-- Using a CTE
WITH EventSportCTE AS (
    SELECT e.event_name, s.sport_name
    FROM dbo.event e
    JOIN dbo.sport s ON e.sport_id = s.id
)
SELECT * FROM EventSportCTE;
```

---

### TSQL12: Using Views and Set Operators

#### ICA Demo 1: Writing Queries using Union Intersect Except set operators

---

Query:

```sql
-- Using UNION to combine competitors from different cities
SELECT full_name, city_id
FROM dbo.games_competitor
WHERE city_id = 1

UNION

SELECT full_name, city_id
FROM dbo.games_competitor
WHERE city_id = 2;
```

---

### ICA Demo 2: More on set operators

---

Query:

```sql
-- Using INTERSECT to find competitors winning gold in both events
SELECT full_name
FROM dbo.competitor_event
WHERE event_id = 1 AND medal_id = 1

INTERSECT

SELECT full_name
FROM dbo.competitor_event
WHERE event_id = 2 AND medal_id = 1;
```

---

### ICA Demo 3: Create inline Table-valued Function

---

Query:

```sql
-- Creating an inline TVF
CREATE FUNCTION GetCompetitorsInCity
(
    @cityId INT
)
RETURNS TABLE
AS
RETURN
(
    SELECT full_name, age
    FROM dbo.games_competitor
    WHERE city_id = @cityId
);

-- Using the inline TVF
SELECT *
FROM dbo.GetCompetitorsInCity(1); -- Replace 1 with the desired city_id
```

---

### TSQL13: Using Window Ranking, Offset, and Aggregate Functions

**ICA TSQL Demo 1 - Partition By Row Number function**

---

Query:

```
-- Using ROW_NUMBER() with PARTITION BY
SELECT
    full_name,
    age,
    city_id,
    ROW_NUMBER() OVER (PARTITION BY city_id ORDER BY age) AS row_num
FROM
    dbo.games_competitor;
```

---

**ICA TSQL Demo 2 - Windows Ranking (or Windows Rank with partition)**

---

Query:

```
-- Using RANK() with PARTITION BY
SELECT
    full_name,
    age,
    city_id,
    RANK() OVER (PARTITION BY city_id ORDER BY age) AS ranking
FROM
    dbo.games_competitor;
```

---

**ICA TSQL Demo 3 - OVER Clause (or CTE Function with Over)**

---

Query:

```sql
-- Using ROW_NUMBER() with OVER clause
SELECT
    full_name,
    age,
    city_id,
    ROW_NUMBER() OVER (PARTITION BY city_id ORDER BY age) AS row_num
FROM
    dbo.games_competitor;
```

---

### ICA TSQL Demo 4 - Writing Aggregate function in Partition By

---

Query:

```sql
-- Using SUM() aggregate function with PARTITION BY
SELECT
    full_name,
    age,
    city_id,
    SUM(age) OVER (PARTITION BY city_id) AS total_age_in_city
FROM
    dbo.games_competitor;
```

---

## TSQL14: Pivoting and Grouping Sets

### ICA Demo 1: Working with Grouping Sets

---

Query:

```sql
-- Using GROUPING SETS for multi-level aggregation
SELECT
    COALESCE(c.city_name, 'Total') AS city_name,
    COALESCE(s.sport_name, 'Total') AS sport_name,
    COUNT(ce.medal_id) AS total_medals
FROM
    dbo.city c
FULL JOIN dbo.games_city gc ON c.id = gc.city_id
FULL JOIN dbo.games g ON gc.games_id = g.id
FULL JOIN dbo.event e ON g.id = e.games_id
FULL JOIN dbo.sport s ON e.sport_id = s.id
FULL JOIN dbo.competitor_event ce ON e.id = ce.event_id
FULL JOIN dbo.medal m ON ce.medal_id = m.id
GROUP BY
    GROUPING SETS (
        (c.city_name, s.sport_name),
        (c.city_name),
        ()
    );
```

#### ICA Demo 2: Writing Queries with PIVOT and UNPIVOT

Query:

```
-- Using PIVOT to transform medal counts by sport
SELECT
    city_name,
    [Basketball] AS Basketball_Medals,
    [Swimming] AS Swimming_Medals,
    [Athletics] AS Athletics_Medals
FROM
    (
        SELECT c.city_name, s.sport_name, COUNT(ce.medal_id) AS medal_count
        FROM dbo.city c
        LEFT JOIN dbo.games_city gc ON c.id = gc.city_id
        LEFT JOIN dbo.games g ON gc.games_id = g.id
        LEFT JOIN dbo.event e ON g.id = e.games_id
        LEFT JOIN dbo.sport s ON e.sport_id = s.id
        LEFT JOIN dbo.competitor_event ce ON e.id = ce.event_id
        GROUP BY c.city_name, s.sport_name
    ) AS MedalCounts
PIVOT
    (
        SUM(medal_count) FOR sport_name IN ([Basketball], [Swimming], [Athletics])
    ) AS PivotTable;
```

# TSQL Part 3: SQL Server Programming

## TSQL15: Executing Stored Procedures

#### ICA Demo 1: T-SQL Stored Procedure

Query:

```
-- Creating a stored procedure to get competitors in a specific city
CREATE PROCEDURE GetCompetitorsInCity

AS
BEGIN
```

```
    SELECT
        full_name,
        age
    FROM
        dbo.games_competitor

END;
```

---

#### ICA Demo 2: TSQL Stored Procedures with Parameters

---

Query:

```

-- Creating a stored procedure with parameters
CREATE PROCEDURE GetCompetitorsBySport
    @sportId INT,
    @minAge INT,
    @maxAge INT
AS
BEGIN
    SELECT
        pc.full_name,
        pc.age,
        s.sport_name
    FROM
        dbo.games_competitor pc
    INNER JOIN
        dbo.event e ON pc.games_id = e.games_id
    INNER JOIN
        dbo.sport s ON e.sport_id = s.id
    WHERE
        s.id = @sportId
        AND pc.age BETWEEN @minAge AND @maxAge;
END;
```

---

### TSQL16: Programming with T-SQL

#### ICA Demo 1: T-SQL programming and Stored Procedure

---

Query:

```

-- Creating a stored procedure
CREATE PROCEDURE GetCompetitorsBySport
    @sportId INT,
    @minAge INT,
    @maxAge INT
```

```
AS
BEGIN
    SELECT
        pc.full_name,
        pc.age,
        s.sport_name
    FROM
        dbo.games_competitor pc
    INNER JOIN
        dbo.event e ON pc.games_id = e.games_id
    INNER JOIN
        dbo.sport s ON e.sport_id = s.id
    WHERE
        s.id = @sportId
        AND pc.age BETWEEN @minAge AND @maxAge;
END;
```

---

#### ICA Demo 2: T-SQL programming with Parameters

---

Query:

```
DECLARE @CityID INT = 1;

SELECT
    full_name,
    age
FROM
    dbo.games_competitor
WHERE
    city_id = @CityID;
```

---

## TSQL Module 17: Implementing Error Handling

#### ICA Demo 1: Implementing T-SQL Error Handling

---

Query:

```
BEGIN TRY
    -- T-SQL code that may cause an error
    DECLARE @Result INT;
    SET @Result = 10 / 0; -- This will cause a divide-by-zero error
END TRY
BEGIN CATCH
```

```
    -- T-SQL code to handle the error
    PRINT 'An error occurred: ' + ERROR_MESSAGE();
END CATCH;
```

---

#### ICA Demo 2: Implementing Structured Exception Handling

---

Query:

```
BEGIN TRY
    -- T-SQL code that may cause an error
    DECLARE @Result INT;
    SET @Result = 10 / 0; -- This will cause a divide-by-zero error
END TRY
BEGIN CATCH
    -- T-SQL code to handle the error
    PRINT 'An error occurred: ' + ERROR_MESSAGE();

    -- You can include additional handling logic here

    -- Optionally, you can rethrow the exception
    THROW;
END CATCH;
```

---

### TSQL Module 18: Implementing Transactions

#### ICA Demo 1: Transactions

---

Starting:

```
BEGIN TRANSACTION;
```

Rollback if you accidently delete production database:

```
ROLLBACK;
```

if you actually update the database correctly:

```
COMMIT;
```

### ICA Demo 2: Controlling Transactions

Query:

```
BEGIN TRANSACTION;

BEGIN TRY
    -- Perform operations within the transaction
    UPDATE dbo.YourTable
    SET Column1 = 'NewValue'
    WHERE Condition;

    -- If everything is successful, commit the transaction
    COMMIT;
END TRY
BEGIN CATCH
    -- If an error occurs, roll back the transaction
    ROLLBACK;

    -- Handle the error or log it
    PRINT 'An error occurred: ' + ERROR_MESSAGE();
END CATCH;
```