

DBMS

Week 05

Triggers

Components

- Event and Condition: Triggers specify when they should be executed, which consists of two parts:
 - Event: The event that triggers the execution of the trigger. Common events include INSERT, UPDATE, DELETE, and other database operations.
 - Condition: The condition that must be satisfied for the trigger to execute. Triggers are executed only if the specified condition is met.
- Actions: Triggers define what actions should be taken when they execute. These actions can include inserting, updating, or deleting data in the database, or even performing more complex operations.

Need for Triggers

- Integrity Constraints: Triggers can enforce integrity constraints that cannot be defined using standard SQL constraints. For example, a trigger can automatically update related data when certain conditions are met, ensuring data consistency.
- Automation: Triggers can automate tasks or alert humans when specific conditions are met. For instance, when the inventory of a product falls below a minimum level, a trigger can automatically generate a reorder request.
- External Actions: While triggers cannot perform direct external actions (e.g., placing orders outside the database), they can prepare data or trigger external processes, such as adding orders to a separate table that is processed by an external system.

Triggers in SQL

- Triggers in SQL are defined using a syntax defined by the SQL standard. Key elements of SQL triggers include:
 - CREATE TRIGGER: Used to define a trigger.
 - AFTER INSERT/UPDATE/DELETE: Specifies when the trigger should execute.
 - FOR EACH ROW: Specifies that the trigger operates on each affected row (row-level trigger).
 - WHEN: Defines the condition that must be satisfied for the trigger to execute.
 - BEGIN...END: Encloses the actions to be performed when the trigger executes.
- SQL triggers can be used to maintain data integrity, automate tasks, and perform various other actions in response to database events.



```
CREATE TABLE Orders (  
    OrderID INT PRIMARY KEY,  
    OrderDate DATETIME,  
    TotalAmount DECIMAL(10, 2)  
);  
GO  
-- Create a trigger that updates OrderDate on INSERT  
CREATE TRIGGER UpdateOrderDate  
ON Orders  
AFTER INSERT  
AS  
BEGIN  
    -- Update OrderDate to the current timestamp  
    UPDATE Orders  
    SET OrderDate = GETDATE()  
    WHERE OrderID IN (SELECT OrderID FROM inserted);  
END;
```

Recursive Queries

Recursive Queries

- In databases, recursive queries are used to deal with hierarchical or interconnected data structures.
- They allow you to traverse and query hierarchical data in a structured and efficient way.
- Recursive queries are often used to find paths between nodes in a tree-like structure or to compute transitive closures in directed graphs.

Transitive Closure

- Transitive closure refers to finding all the elements that are indirectly related to a given element through a sequence of direct relationships.
- In the context of databases, it is commonly used to find all the indirect prerequisites or dependencies of a specific item or entity.
- For example, finding all the courses that are prerequisites, either directly or indirectly, for a particular course.

Iterative Approach


- One way to compute transitive closure is through iteration.
- You start with the direct relationships, then iteratively add indirect relationships until there are no more to add.
- In your example, the function `findAllPrereqs(cid)` demonstrates this approach.
- It uses temporary tables to keep track of courses at different levels of prerequisites until there are no new courses to add.

Recursive Approach

- A more elegant and efficient way to compute transitive closure is by using recursive queries.
- In SQL, recursive queries are defined using the WITH RECURSIVE clause.
- You start with the base case (direct relationships) and then define how to find the next level of relationships based on the current level.
- The database engine takes care of the recursion for you.
- For example, in your SQL query, WITH RECURSIVE rec_prereq(course_id, prereq_id) defines a recursive view that finds all courses that are prerequisites either directly or indirectly.



```
CREATE TABLE CoursePrerequisites (  
    CourseID VARCHAR(8) PRIMARY KEY,  
    PrerequisiteID VARCHAR(8)  
);  
GO  
INSERT INTO CoursePrerequisites (CourseID, PrerequisiteID)  
VALUES  
    ('CS-101', NULL), -- CS-101 has no prerequisites  
    ('CS-105', 'CS-101'),  
    ('CS-107', 'CS-101'),  
    ('CS-201', 'CS-105'),  
    ('CS-202', 'CS-105'),  
    ('CS-203', 'CS-107'),  
    ('CS-301', 'CS-202'),  
    ('CS-302', 'CS-203'),  
    ('CS-303', 'CS-301');
```



```
WITH RecursivePrerequisites AS (  
    SELECT CourseID, PrerequisiteID  
    FROM CoursePrerequisites  
    WHERE CourseID = 'CS-301' -- Starting course  
    UNION ALL  
    SELECT cp.CourseID, cp.PrerequisiteID  
    FROM CoursePrerequisites cp  
    INNER JOIN RecursivePrerequisites rp ON cp.CourseID = rp.PrerequisiteID  
)  
SELECT DISTINCT CourseID AS Prerequisite  
FROM RecursivePrerequisites  
WHERE CourseID <> 'CS-301'; -- Exclude the starting course
```

Output
Prerequisite

CS-101
CS-105
CS-201
CS-202

Advanced Aggregation Features

Ranking

- Ranking is used to find the position of a value within a set. SQL provides the `RANK()` and `DENSE_RANK()` functions to assign ranks to rows based on specific criteria.
- It's often used for scenarios like ranking students by their GPA.



```
SELECT student_id, RANK( ) OVER (ORDER BY gpa DESC) AS gpa_rank  
FROM students;
```


Windowing

- Windowing allows you to compute aggregate functions over specific ranges or windows of rows.
- This is useful for tasks like trend analysis. You can specify the window using clauses like `ROWS BETWEEN ... AND`



```
SELECT date, revenue, AVG(revenue) OVER (ORDER BY date ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS  
moving_avg  
FROM sales;
```

Pivoting

- Pivoting is a way to transform rows into columns, creating cross-tabulations or pivot tables.
- SQL provides the PIVOT clause to achieve this, allowing you to summarize data across multiple dimensions.



```
SELECT item_name,  
       SUM(CASE WHEN color = 'dark' THEN quantity ELSE 0 END) AS dark,  
       SUM(CASE WHEN color = 'pastel' THEN quantity ELSE 0 END) AS pastel,  
       SUM(CASE WHEN color = 'white' THEN quantity ELSE 0 END) AS white  
FROM sales  
GROUP BY item_name;
```

Rollup

- The ROLLUP operation is used for creating multiple levels of subtotals and grand totals in a single query.
- It generates groupings based on the specified columns and produces results at various levels of aggregation.



```
SELECT item_name, color, clothes_size, SUM(quantity) AS total_quantity  
FROM sales  
GROUP BY ROLLUP (item_name, color, clothes_size);
```

Cube

- The CUBE operation is like ROLLUP, but it generates all possible combinations of groupings for the specified columns.
- It's useful for multidimensional analysis.



```
SELECT item_name, color, SUM(quantity) AS total_quantity  
FROM sales  
GROUP BY CUBE (item_name, color);
```


Grouping Sets

- While ROLLUP and CUBE generate predefined groupings, you can have more control over the groupings using the GROUPING SETS construct.
- It allows you to specify exactly which combinations of columns to include in the grouping.



```
SELECT item_name, color, SUM(quantity) AS total_quantity  
FROM sales  
GROUP BY GROUPING SETS ((item_name), (color), ());
```

Class activity

- JDBC
- Prepared statements
- SQL injection
- Metadata
- Updatable result sets
- Open Database Connectivity (ODBC)
- Embedded SQL
- Embedded database
- Stored procedures and functions
- Table functions.
- Parameterized views
- Persistent Storage Module (PSM).
- Exception conditions
- Handlers
- External language routines
- Sandbox
- Trigger
- Transitive closure
- Hierarchies
- Create temporary table
- Base query
- Recursive query
- Fixed point
- Monotonic
- Windowing
- Ranking functions
- Cross-tabulation
- Cross-tab
- Pivot-table
- Pivot
- SQL group by cube, group by rollup

Part three

Database Design

Chapter 6 database design using e-r model

Database design using the Entity-Relationship (E-R) model involves creating a blueprint for how data will be organized, stored, and related in a relational database. The E-R model helps you define the entities (objects), their attributes (properties), and the relationships between them.

Overview of the Design Process

Design Phases

- **Characterize User Requirements:** In this initial phase, the goal is to fully understand the data needs of the prospective users of the database. This often requires extensive interaction with domain experts and users to gather requirements.
- **Choose a Data Model:** The next step is to select an appropriate data model that will serve as the basis for the database design. Common models include the entity-relationship model (ER), relational model, and more.
- **Conceptual Schema:** In this phase, the high-level requirements are translated into a conceptual schema. The schema defines entities, attributes, relationships, and constraints. Entity-relationship diagrams are often used for graphical representation.
- **Functional Requirements:** Alongside the conceptual schema, the functional requirements of the enterprise are identified. This includes specifying the types of operations (transactions) that will be performed on the data.
- **Logical Design:** The logical-design phase involves mapping the conceptual schema to the specific data model of the chosen database system. For many systems, this means mapping to the relational model.
- **Physical Design:** In the final phase, the physical schema of the database is defined. This includes specifying file organization, indexing structures, and other physical storage details.

Design Alternatives

- During the design process, the database designer needs to make choices and decisions to avoid common pitfalls:
 - Redundancy: Redundancy should be minimized to avoid inconsistencies. Data should ideally be stored in one place, and updates should be carefully managed to maintain consistency.
 - Incompleteness: The design should not make it difficult to model certain aspects of the enterprise. Missing entities or relationships can hinder the ability to represent the full scope of data requirements

Challenges

- Database design can be challenging due to the complexity of real-world applications.
- There are often many potential design choices, and the designer must carefully consider how to represent entities, attributes, relationships, and constraints.
- Making choices like whether a relationship should be represented as an entity or as a direct relationship between entities can have significant implications for the database's ability to accurately model the enterprise.

Good Design Principles

- A combination of scientific principles and good design practices is required to create an effective database schema.
- The goal is to represent the data requirements accurately, minimize redundancy, and avoid design flaws that could lead to inconsistencies or limitations in representing the enterprise's data needs.