Active Databases Part 1: Introduction

CS561

Active Databases

- **Triggers and rules are developed for data integrity and constraints**

- **Triggers make “passive” database “active”**
  - Database reacts to certain situations

- **Event Condition Action rule :**
  - on event insert/update/delete,
  - if condition C is true
  - then do action A

Brief History

- 1975: Idea of "integrity constraints"
- Mid 1980-1990: research in constraints & triggers
  - Languages and algorithms
- SQL-92: constraints
  - Key constraints; referential integrity, domain constraints
  - Declarative spec and procedural interpretation
- SQL-99: triggers/ECA (limited)
  - Early acceptance; Widely-varying support in products; "execution semantics" differ, how far to go ?

Event-Condition-Action (ECA)

- Event occurs in databases
  - addition of new row, deletion of row by DBMS
- Conditions are checked
  - SQL condition
- Actions are executed if conditions are satisfied
  - SQL + procedures
  - All data actions performed by the trigger execute within the same transaction in which the trigger fires,

Triggers

- A procedure that runs automatically when a certain event occurs in the DBMS

- Implementation of the ECA rules

- **The procedure performs some actions, e.g.,**
  - Check certain values
  - Fill in some values
  - Inserts/deletes/updates other records
  - Check that some business constraints are satisfied
  - Cancel or roll back forbidden actions

Database Triggers in SQL

- Available in most enterprise DBMSs (Oracle, IBM DB2, MS SQL server) and some public domain DBMSs (Postgres)

- Some vendor DBMS permit native extensions to SQL for specifying the triggers
  - e.g. PL/SQL in Oracle, Transact SQL in MS SQL Server

- Some DBMS extend the triggers beyond tables
  - for example also to views as in Oracle

Trigger Components

- **Three components**
  - **Event:** When this event happens, the trigger is activated
  - **Condition (optional):** If the condition is true, the trigger executes, otherwise skipped
  - **Action:** The actions performed by the trigger

- **Semantics**
  - When the **Event** occurs and **Condition** is true, execute the **Action**

Types of SQL Triggers

- How many times should the trigger body execute when the triggering event takes place?
  - **Per statement:** the trigger body executes once for the triggering event. This is the default.
  - **For each row:** the trigger body executes once for each row affected by the triggering event.

- When the trigger can be fired
  - Relative to the execution of an SQL DML statement (**before** or **after** or instead of it)
  - Exactly in a situation depending on specific system resources (e.g. signal from system clock)

Statement and Row Triggers

Example 1: Monitoring Statement Events

```
SQL> INSERT INTO dept (deptno, dname, loc)
  2  VALUES (50, 'EDUCATION', 'NEW YORK');
```

Execute only once even if multiple rows affected

Example 2: Monitoring Row Events

```
SQL> UPDATE emp
  2  SET sal = sal * 1.1
  3  WHERE deptno = 30;
```

Execute for each row of table affected by event

Granularity of Event

- An UPDATE or DELETE statement may update (or delete) many records at the same time
  - May insert many records in the same statement as well

- **Does the trigger execute for each updated or deleted record, or once for the entire statement ?**
  - **We define such granularity**

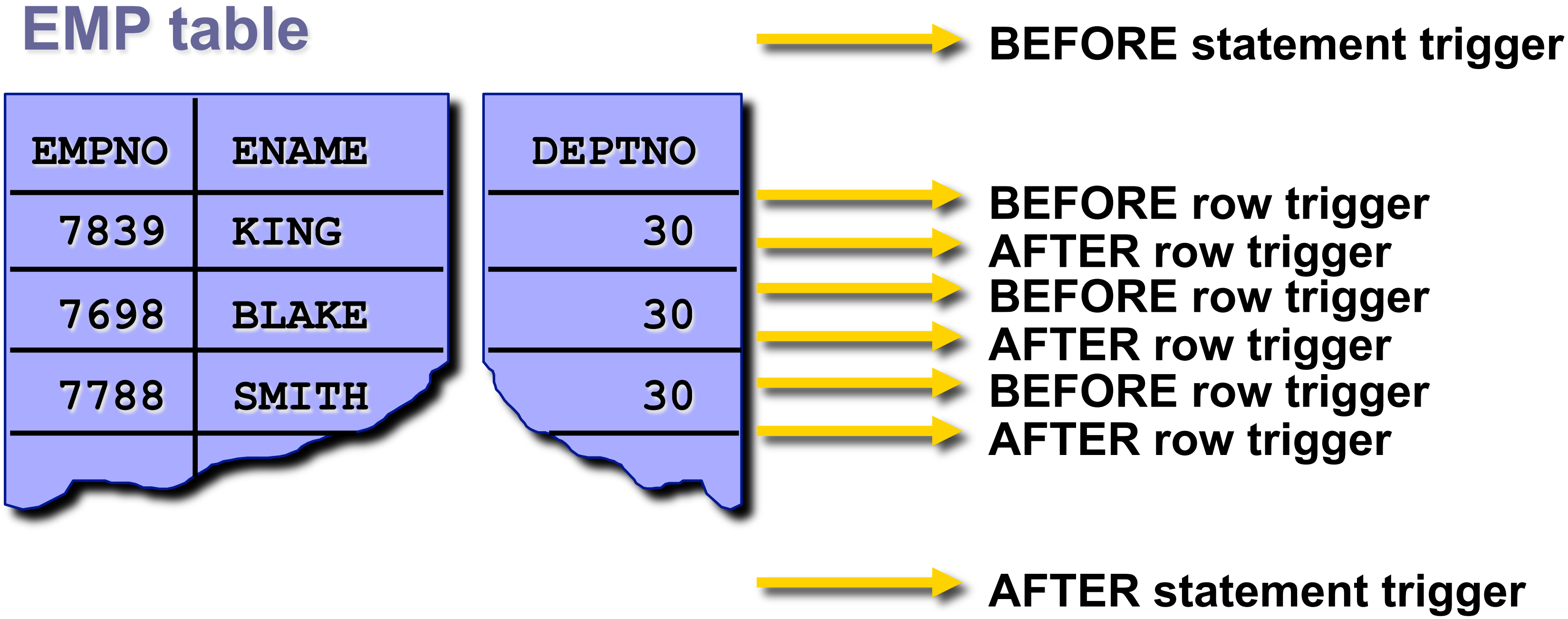
```
Create Trigger <name>
Before| After       Insert| Update| Delete

For Each Row | For Each Statement
….
```

That is the timing

That is the event

That is the granularity

Firing Sequence of Database Triggers on Multiple Rows



EMP table

EMPNO	ENAME	DEPTNO
7839	KING	30
7698	BLAKE	30
7788	SMITH	30

BEFORE statement trigger

BEFORE row trigger
AFTER row trigger
BEFORE row trigger
AFTER row trigger
BEFORE row trigger
AFTER row trigger

AFTER statement trigger

Example: Logging Operations

```
SQL> CREATE TRIGGER increase_salary_trg
  2          AFTER UPDATE OF sal
  3          ON emp
  4   BEGIN
       if :new.sal > :old.sal Then
  5      INSERT INTO sal_hist(increased, changedOn)
  6          VALUES (‘YES’, SYSDATE);
       end;
  7   END;
  8   /
```

Trigger name:	`increase_salary_trg`
Timing:	`AFTER` executing the statement
Triggering event:	UPDATE of `sal` column
Target:	`emp` table
Trigger action:	INSERT values INTO `sal_hist table`

Example: Checking Values

If the employee salary increased by more than 10%, make sure the ‘comment’ field is not empty and its value has changed, otherwise reject the update

```
Create Trigger EmpSal
Before Update On Employee
Referencing
            OLD ROW AS   oldRec,
           NEW ROW AS  newRec
For Each Row
Begin
     IF (newRec.salary > oldRec.salary * 1.10) Then
        IF (newRec.comment = ‘’ or newRec.comment is null or
            newRec.comment = oldRec.comment)
             RAISE_APPLICATION_ERROR(-20004, ‘Comment field not correct’);
       End IF;
     End IF;
End;
```

Example: Using Temp Variable

If the newly inserted record in employee has null date field, fill it in with the current date

```
Create Trigger EmpDate
Before Insert On Employee
Referencing
            NEW ROW AS  newRec
For Each Row
Declare
    temp date;
Begin
     Select sysdate into temp from dual;
     IF (newRec.date is null) Then
         newRec.date := temp;
    End IF;
End;
```

Define variables

Oracle system table always has the current date

Updating the new value to be inserted

Example: Calculating Derived Columns

```
SQL>CREATE OR REPLACE TRIGGER derive_commission_trg
  2 BEFORE UPDATE OF sal ON emp
  3 FOR EACH ROW
  4 WHEN (new.job = 'SALESMAN')
  5 BEGIN
  6    :new.comm := :old.comm * (:new.sal/:old.sal);
  7 END;
  8 /
```

Trigger name:	derive_commission_trg
Timing:	BEFORE executing the statement
Triggering event:	UPDATE of sal column
Filtering condition:	job = ‘SALESMAN’
Target:	emp table
Trigger parameters:	old, new
Trigger action:	calculate the new commission to be updated

Controlling Triggers using SQL

- Disable/Re-enable database trigger

```
ALTER TRIGGER trigger_name  DISABLE | ENABLE
```

- Disable or Re-enable all triggers for table

```
ALTER TABLE table_name   DISABLE | ENABLE  ALL TRIGGERS
```

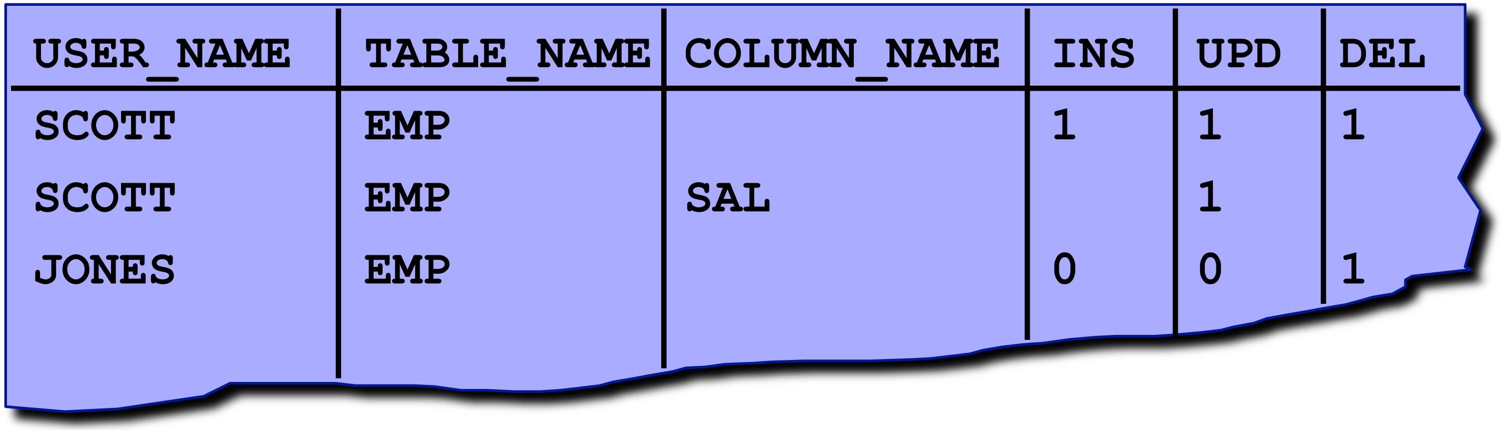
- Removing a trigger from database

```
DROP TRIGGER trigger_name
```

Using Database Triggers

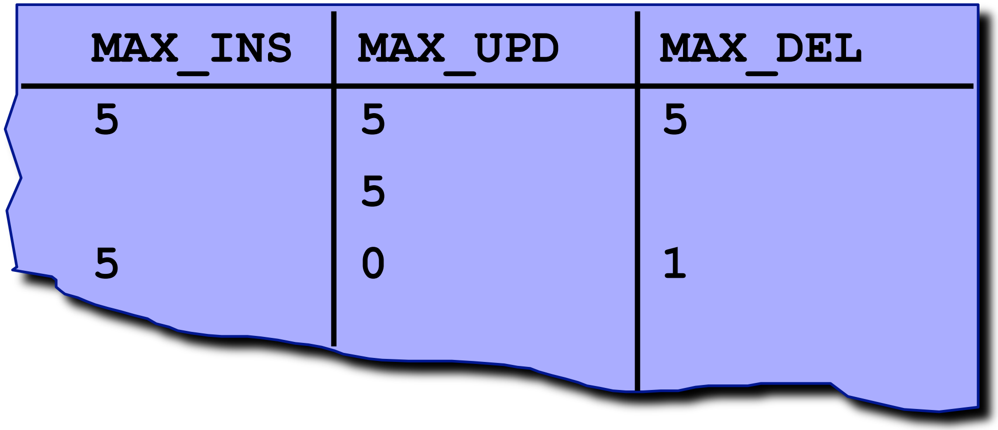
- **Auditing Table Operations**
  - each time a table is updated auditing information is recorded against it
- **Tracking Record Value Changes**
  - each time a record value is changed the previous value is recorded
- **Maintenance of Semantic Integrity**
  - **e.g.** when the factory is closed, all employees should become unemployed
- **Storing Derived Data**
  - **e.g.** the number of items in the trolley should correspond to the current session selection
- **Security Access Control**
  - **e.g.** checking user privileges when accessing sensitive information

Auditing Table Operations



USER_NAME	TABLE_NAME	COLUMN_NAME	INS	UPD	DEL
SCOTT	EMP		1	1	1
SCOTT	EMP	SAL		1	
JONES	EMP		0	0	1

… continuation



MAX_INS	MAX_UPD	MAX_DEL
5	5	5
	5	
5	0	1

Example: Counting Statement Execution

```
SQL>CREATE OR REPLACE TRIGGER audit_emp
  2 AFTER DELETE ON emp
  3 FOR EACH ROW
  4 BEGIN
  5      UPDATE audit_table SET del = del + 1
  6      WHERE user_name = USER
  7      AND table_name = 'EMP’;
  7 END;
  8 /
```

Whenever an employee record is deleted from database, counter in an audit table registering the number of deleted rows for current user in system variable USER is incremented.

Example: Tracing Record Value Changes

USER_NAME	TIMESTAMP	ID	OLD_LAST_NAME	NEW_LAST_NAME
EGRAVINA	12-SEP-04	7950	NULL	HUTTON
NGREENBE	10-AUG-04	7844	MAGEE	TURNER

… continuation

OLD_TITL E	NEW_TITLE	OLD_SALARY	NEW_SALARY
NULL	ANALYST	NULL	3500
CLERK	SALESMAN	1100	1100

Example: Recording Changes

```
SQL>CREATE OR REPLACE TRIGGER audit_emp_values
  2 AFTER UPDATE ON emp
  3 FOR EACH ROW
  4 BEGIN
  5   INSERT INTO audit_emp_values (user_name,
  6    timestamp, id, old_last_name, new_last_name,
  7    old_title, new_title, old_salary, new_salary)
  8   VALUES (USER, SYSDATE, :old.empno, :old.ename,
  9    :new.ename, :old.job, :new.job,
 10    :old.sal, :new.sal);
 11 END;
 12 /
```

Whenever some details for an employee are updated, both the previous and new details are recorded in an audit table to allow tracing the history of changes. An insert operation cannot be recorded with this trigger as old.empno has no value.

Restrictions for Database Triggers

- **Problem**: impossible to determine certain values during execution of a sequence of operations belonging to one and the same transaction

- **Mutating tables**: contain rows which change their values after certain operation and which are used again before the current transaction commits

Example: Mutating Table

```
SQL> CREATE OR REPLACE TRIGGER emp_count
  2  AFTER DELETE ON emp
  3  FOR EACH ROW
  4  DECLARE
  5      num INTEGER;
  6  BEGIN
  7      SELECT COUNT(*) INTO num FROM emp;
  8      DBMS_OUTPUT.PUT_LINE(' There are now ' ||
        num || ' employees.');
  9  END;
 10  /
```

```
SQL> DELETE FROM emp
  2  WHERE  deptno = 30;

ERROR at line 1:
ORA-04091: table CGMA2.EMP is mutating, trigger/
function may not see it
```

Under the bar is code entered in SQL-PLUS which triggers cascade_updates in this case. Triggers are not executed directly.

Example: Mutating Table (fixed)

```
SQL> CREATE OR REPLACE TRIGGER emp_count
  2  AFTER DELETE ON emp
  3  -- FOR EACH ROW
  4  DECLARE
  5      num INTEGER;
  6  BEGIN
  7      SELECT COUNT(*) INTO num FROM emp;
  8      DBMS_OUTPUT.PUT_LINE(' There are now ' ||
         num || ' employees.');
  9  END;
 10  /
```

Now the trigger becomes a statement trigger and the EMP table is no longer mutating.

```
SQL> DELETE FROM emp  WHERE  deptno = 30;

There are now 8 employees.

6 rows deleted.
```

Summary

- **Triggers change databases from “passive” to “active”**

- **Triggers have Event-Condition-Action**
  - Event: I/U/D
  - Timing: Before/After
  - Granularity: Row-level/Statement-level

- **Usage:**
  - Auditing Table Operations
  - Tracking Record Value Changes
  - Maintenance of Semantic Integrity
  - Storing Derived Data
  - Security Access Control

Active Databases Part 2: Classifications & Scalability

CS561

Types of Triggers

- **Generated**: based on some higher-level specification
  - Foreign keys, primary keys, unique constraints, etc.
- **Handcrafted**: usually specific to some application
  - Capture the application semantics

Why “Generated” Triggers

- Triggers (active rules) are difficult to write correctly

- Idea:
  - Trigger application specified at higher level (declarative)
  - Automatic generation of actual triggers
  - Guaranteed Correctness

Classification of Usage

- **Generated Triggers**
  - □ **Kernel DBMS:** hard coded into kernel
  - □ **DBMS services:** enhances database functionality
  - □ **External applications:** creating triggers specific to application

- **Handcrafted Triggers**
  - □ **External applications:** creating triggers specific to application

“Generated” Triggers/ DBMS Kernel

- **Referential integrity**
  - If foreign key in a table is deleted or updated, it causes an action usually specified by user: set null/cascade
  - Primary keys, Unique columns, etc…
- **Materialized views**
  - Set of triggers that keep data consistent
    - Either re-computes view, or
    - Better changes view each time base data is changed

“Generated” Triggers/ DBMS Services

- Alerter
  - When data changes, message can be sent to user
- Replication
  - If a table is copied, a trigger will observe updates to that original table and will change copied table.
- Audit Trails
  - Monitoring any changes over a given table

“Generated” Triggers/ External Applications

- Workflow management
- External tools with support for generation of “Process Rules/Models”

“Handcrafted” Triggers

- External Applications
  - Straightforward use of triggers
  - Application specific
    - Additional forms of “data integrity”
    - Could be used to compute derived columns
    - Or, enforce arbitrarily complex application-specific semantics

- Examples:
  - Business rules, supply chain management, web applications, etc.

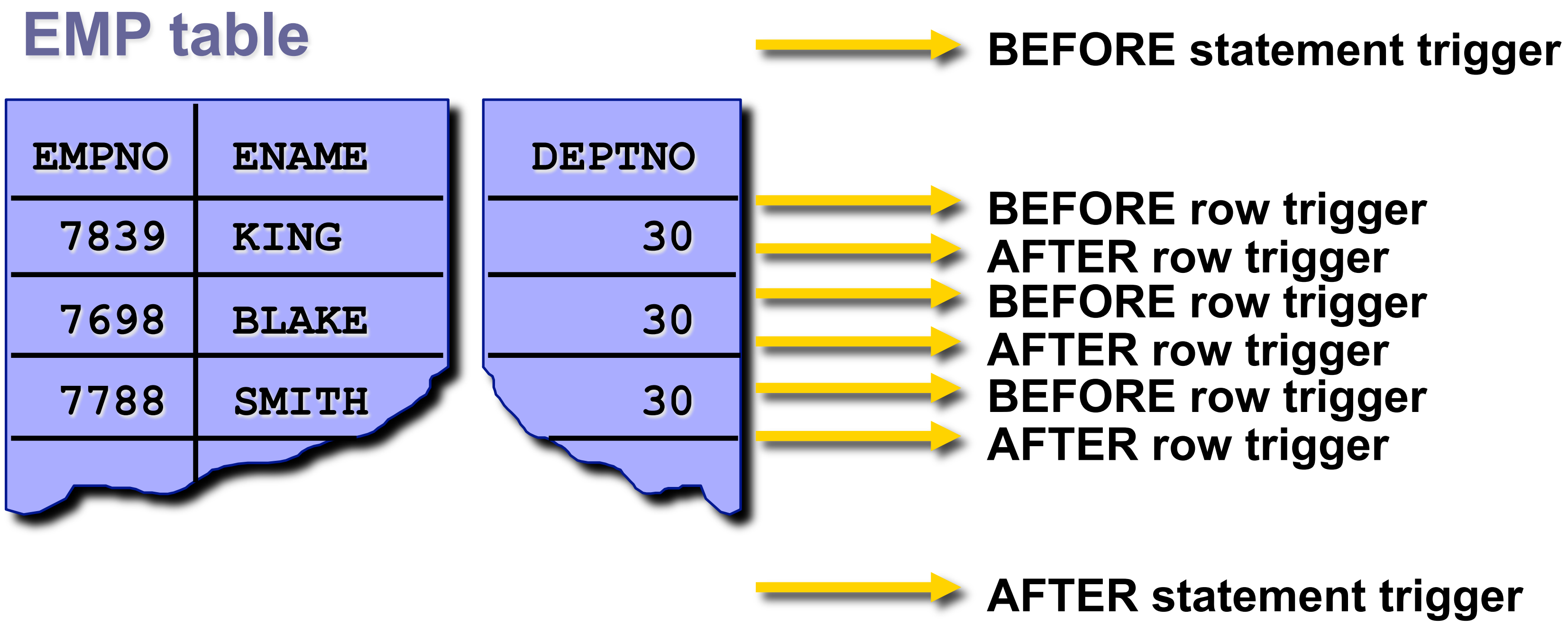
Challenges

Challenge : Semantics ?

- What causes a rule to be triggered? (states, ops, transitions)
- At what granularity are rules triggered ? (after tuple change, set level change, transaction, etc).
- What happens when multiples rules are triggered? (arbitrary order, numeric or priorities suggested)
- Can rules trigger each other, or themselves?

In general, many subtle design choices exist !

Multiple Triggers at Same Event



EMP table

EMPNO	ENAME	DEPTNO
7839	KING	30
7698	BLAKE	30
7788	SMITH	30

BEFORE statement trigger

BEFORE row trigger
AFTER row trigger
BEFORE row trigger
AFTER row trigger
BEFORE row trigger
AFTER row trigger

AFTER statement trigger

Challenge: Rule Analysis

- **Termination:** produces a final state

- **Confluence :** terminates and produces a final state that does not depend on order of execution

- **Termination :**
  - □ Find cycles
  - □ Examine rules in cycle for behavior
  - □ Could determine that terminate in some cases
  - □ Data dependent : even if at compile-time has cycle, still may be useful

- **In practice (Oracle) :**
  - □ Optimistic solution
  - □ Terminate after 25 trigger invocations, and rollback

Scalable Trigger Processing

Discussion of publication by

Eric N. Hanson et al

Int Conf Data Engineering 1999

CS561

Motivation

- Triggers popular for:
  - Integrity constraint checking
  - Alerting, logging, etc.

- Commercial database systems
  - Limited triggering capabilities
  - few trigger/update-type on table; or at best 100.

- **But : Current technology doesn‘t scale well**
- **And, internet and web-based applications may need millions of triggers.**

Problem Definition

- Given: Relational DB, Trigger statements, Data Stream
- Find: Triggers corresponding to each stream item
- Objective: Scalable trigger processing system

- **Assumptions:**
  - Number of distinct structures of trigger expressions is relatively small
  - All trigger expression structures small enough to fit in main memory

Overall Driving Idea

- If large number of triggers are created, then many have the same format.
- Triggers share same expression signature except that parameters substituted.
- Group predicates from trigger conditions based on expression signatures into equivalence classes
- Store them in efficient main memory data structures

Triggers for stock ticker notification

- **Create trigger** *T1* **from** *stock*
  when *stock.ticker* = ‘GOOG’ and *stock.value* < 500
  do *notify_person(P1)*

- **Create trigger** *T2* **from** *stock*
  when *stock.ticker* = ‘MSFT’ and *stock.value* < 30
  do *notify_person(P2)*

- **Create trigger** *T3* **from** *stock*
  when *stock.ticker* = ‘ORCL’ and *stock.value* < 20
  do *notify_person(P3)*

- **Create trigger** *T4* **from** *stock*
  when *stock.ticker* = ‘GOOG’
  do *notify_person(P4)*

Expression Signature

- Idea: Common structures in condition of triggers
- *T1: stock.ticker* = ‘GOOG’ and *stock.value* < 500
  T2: *stock.ticker* = ‘MSFT’ and *stock.value* < 30
  T3: *stock.ticker* = ‘ORCL’ and *stock.value* < 20

  Expression Signature:
    - *E1: stock.ticker* = ***const1*** and *stock.value* < ***const2***
- *T4: stock.ticker* = ‘GOOG’
  - □ Expression Signature:
    - *E2: stock.ticker* = ***const3***
- Expression signature defines equivalence class of all instantiations of expression with different constants

Main Idea

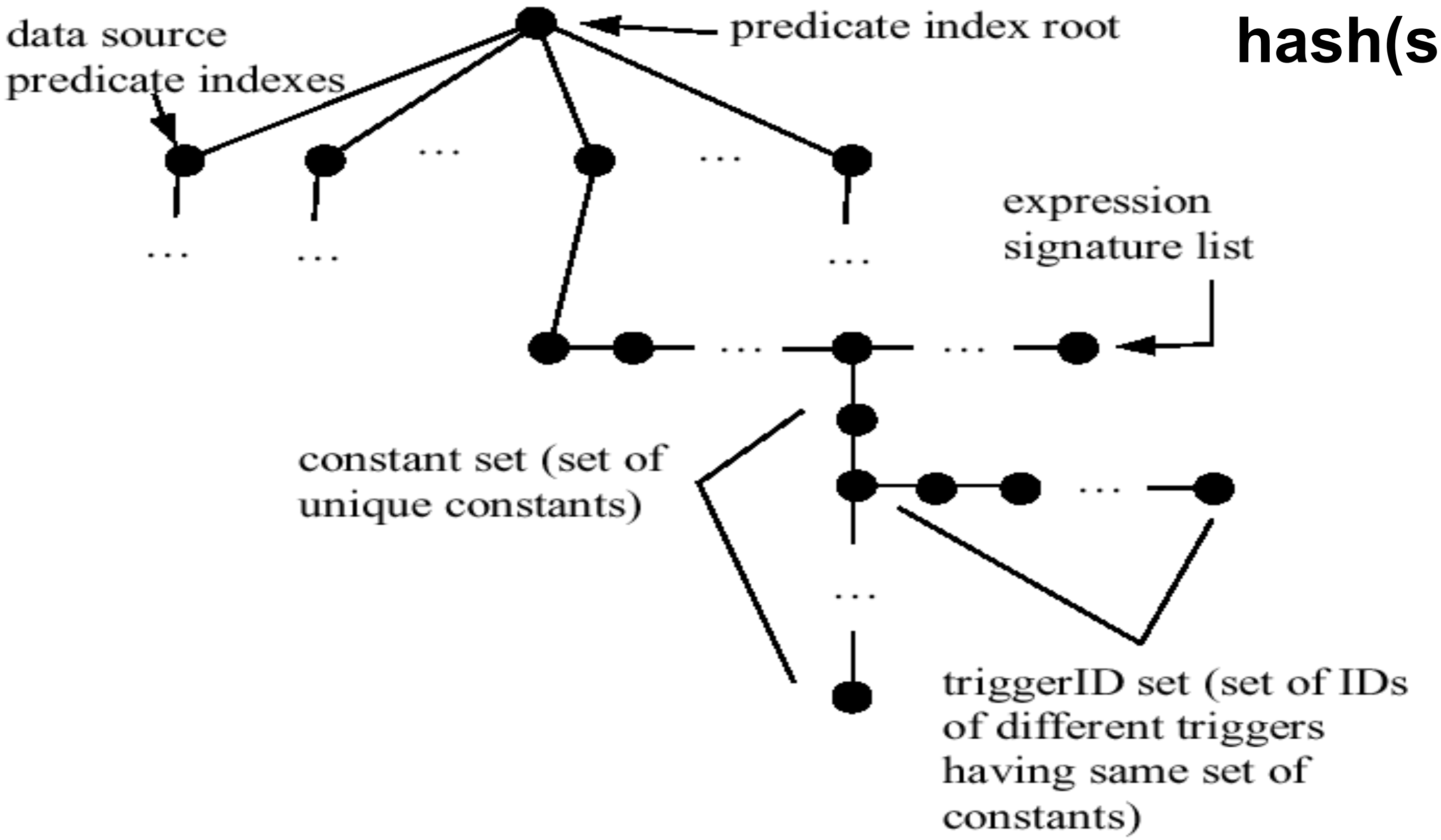
- Only a few distinct expression signatures, build data structures to represent them explicitly (in memory)

- Create constant tables that store all different constants, and link them to their expression signature

Main Structures

- A-treat Network
  - Network for trigger condition testing
    - For a trigger to fire, all conditions must be true
- Expression Signature
  - Common structure in a trigger
    - *E1: stock.ticker* = *const1* and *stock.value* < *const2*
- Constant Tables
  - Constants for each expression signature

Predicate Index



hash(src-ID)

Goal: Given an update, identify all predicates that match it.