Integrity Constraint Management

CS2312

The correctness and consistency of the data and its information

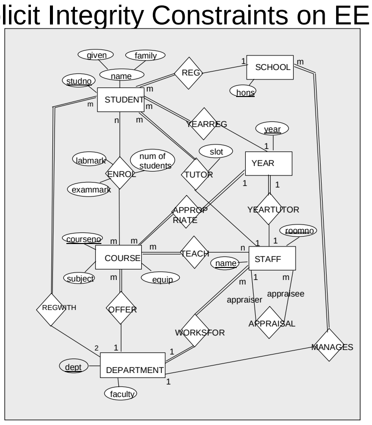
- Implicit
  - of the data model
  - specified and represented in schema
- Explicit
  - additional constraints of world
  - can't directly represent in data model
- Inherent
  - assumed to hold by the definition of the data model
  - don't have to be specified
  - e.g. attribute is *atomic*

Classification of constraints

- State constraints
  - Constraints on the database state
  - State is consistent if it satisfies all the state constraints
- Transition constraints
  - Constraint on the transition from one state to another, not an individual state
  - e.g. labmark of a student can only be increased
  - ∴ need to know the new value of labmark and the old value of labmark

    newlabmark >= oldlabmark

Explicit Integrity Constraints on EER Model



Explicit Integrity Constraints on EER Model

1.Student's tutor must be employed by a department that the student is registered with

2. A student can only be enrolled for a course which is appropriate to the year that the student is in

3. Only staff who are employed by a department can teach a course offered by the department

4. Staff can only be appraised by a member of staff in the same department

5. Staff who don't lecture must tutor

6. Average mark for a course > 30

7. Labmarks can only increase

REGWITH can be represented by either

a) STUDENT(<u>studno</u>, familyname, givenname, hons, tutor, slot, dept1, dept2) or

b) REGWITH(<u>studno, dept</u>)

Classification of state integrity constraints

- *Uniqueness:* no two values of the same attribute can be equal
- *Entity Integrity:* no part of the key of a relation can be null
- *Non-Null:* all values of an attribute must be non-null
- *Domains (value sets):* all values of an attribute must lie within a defined domain, e.g. 0 < x < 100
- *Inter-domain matches:* would not be sensible to match disparate domains
- *Domain cardinality:* the number of values for an attribute must lie in a defined range , e.g. number of natural parents living: 0, 1 or 2

Revision … Revision … Revision …

Classification of state integrity constraints

- *Relationship cardinality :* the number of times an entity instance can participate in a relationship instance

e.g. a student can take many courses and a course can be taken by many students; students can only enrol for up to 5 courses.

- *Relationship participation:* entity instances can optionally or mandatorally participate with a relationship instance

e.g. A child must mandatorally be related through a mother relationship to a person but a person can be optionally related to a child

Revision ... Revision ... Revision ...

Classification of state integrity constraints

- *Inclusion:* all values of one attribute are also values of another

e.g. set of appraisers ⊂ set of staff
set of undergraduates ⊂ set of students

- *Covering:* all values of one attribute are also values of one of a set of attributes

e.g. cars ∪ boats ∪ planes = vehicles
undergraduates ∪ postgraduates = students

- *Disjointedness:* the value of an attribute cannot be at the same time for a particular entity more than one value e.g. male and female
- *Referential:* a value under one attribute is guaranteed to exist if there is a corresponding value under another attribute;

e.g. every student's tutor attribute must match a staff entity

Revision ... Revision ... Revision ...

General

- More general constraints consisting of a predicate over values under an attribute or across attributes.
- Sometimes known as business rules
- *Inter-attribute constraints*
  - date of birth < date of entry
  - quantity ordered = quantity delivered
- *Domain set functions*
  - average mark of students > 30
- *Derived attributes*
- number of students enrolled on a course = $_{\text{studno}}\, ƒ\, _{\text{COUNT courseno}}$ (ENROL)
- total mark for a course = exammark + labmark

Specifying Constraints in the Relational Model

- Inherent
  - already in model

  e.g. atomic domain values
- Implicit
  - in the Data Definition Language

  e.g. referential integrity
- Explicit
  - Declaratively *assertions or triggers*
  - Procedurally *transactions*

  e.g. year tutors supervise two fewer students than other staff

Domain integrity in SQL2

```
Create domain name_type as char(20);
create table student
   (studentno number(8) primary key,
   givenname name_type,
   surname name_type,
   hons char(30) check (hons in
   ('cis','cs','ca','pc','cm','mcs')),
   tutorid number(4),
   yearno number(1) not null,    etc.....
```

```
create table staff
(staffid number(4) primary key,
givenname name_type,
surname name_type,
title char(4)
check (title in ('mrs','mr','ms','prof','rdr','dr')),
roomno char(6),
appraiserid number(4),         etc....
```

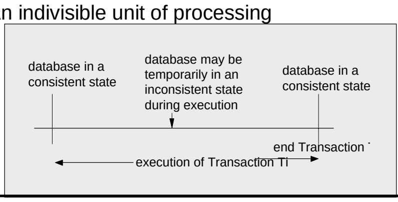
Extensions to Referential Integrity in SQL2

```
create table YEAR
   (yearno number(8),
   yeartutorid number(4) constraint fk_tut
      references STAFF(staffid)
      on delete set null on update cascade),
   constraint year_pk1 primary key (yearno));

create table STAFF
   (staffid number(4) primary key,
   givenname char(20),
   surname char(20),
   title char(4)
   check (title in    ('mrs','mr','ms','prof','rdr','dr')),
   roomno char(6),
   appraiserid number(4) not null default '22',
   constraint app_fk
      foreign key (appraiserid)
      references STAFF(staffid) disable
      on delete set default on update cascade);
```

Controlled redundancy in Transactions

- An atomic (all or nothing) program unit that performs database accesses or updates, taking a consistent (& correct) database state into another consistent (& correct) database state
- A collection of actions that make consistent transformations of system states while preserving system consistency
- An indivisible unit of processing



Controlled redundancy in Transactions

- STUDENT(<u>studno</u>, name, numofcourses)
  COURSE(<u>courseno</u>,subject,numofstudents)
  ENROL(<u>studno,courseno</u>)
- *Students can only enrol for up to 5 Courses.*
- *Add student S to course C*

1. select course C
2. select student S
3. count number of courses S already enrolled for
   if < 5 then step 4   if = 5 then halt
   END
4. select enrol for student S
5. check whether S already enrolled on C
   if no then step 6   if yes then halt
   END
6. Insert enrol instance (S,C)
7. Increment numofcourses in student for S
8. Increment numofstudents in course for C

END

Constraints Managed Procedurally

- Problems:
  - load on programmer
  - changing constraints
  - no centralised enforcement
  - no central record
- In Oracle, transactions written in host programming languages (e.g. C) or PL/SQL
- PL/SQL programs can be saved in the Data Dictionary as
  - Functions
  - Procedures
  - Packages

Database Triggers

- Centralized actions can be defined using a non declarative approach (writing PL/SQL code) with database triggers.
- A database trigger is a stored procedure that is fired (implicitly executed) when an INSERT, UPDATE, or DELETE statement is issued against the associated table.
- Database triggers can be used to customize a database management system:
  - value-based auditing
  - automated data generation
  - the enforcement of complex security checks
  - enforce integrity rules
  - enforce complex business rules

Trigger Structure

A trigger has three basic parts:

- Event
  - a triggering ***event*** or statement
    - the SQL statement that causes a trigger to be fired
- Condition
  - a trigger restriction or ***condition***
    - specifies a Boolean expression that must be TRUE for the trigger to fire. The trigger action is not executed if the trigger restriction evaluates to FALSE or UNKNOWN.
- Action
  - a trigger ***action***
    - the procedure (PL/SQL block) that contains the SQL statements and PL/SQL code to be executed when a triggering statement is issued and the trigger restriction evaluates to TRUE.

Example : maintaining derived values

Event

```
CREATE OR REPLACE TRIGGER increment_courses
 AFTER INSERT ON enrol
```

Condition

Action

row trigger

column values for current row and new/old correlation names

```
FOR EACH ROW
BEGIN
update students
  set numofcourses = numofcourses + 1
  where students.studno = :new.studno
END;
```

SQL statements

Example Integrity Trigger in Oracle

Event

Condition

Action

```
 CREATE TRIGGER labmark_check
 BEFORE INSERT OR UPDATE OF labmark ON enrol
DECLARE
  bad_value exception;
WHEN (old.labmark IS NOT NULL OR new.labmark IS
  NOT NULL)
FOR EACH ROW
BEGIN
   IF :new.labmark < :old.labmark
   THEN raise bad_value ;
   END IF;
   EXCEPTION
     WHEN bad_value THEN
          raise_application_error(-20221,‘New
     labmark lower         than old labmark’ );
END;
```

row trigger

column values for current row and new/old correlation names

SQL and PL/SQL statements, PL/SQL language constructs (variables, constants, cursors, exceptions etc), and call stored procedures.

Example Reorder Trigger in Oracle

```
CREATE TRIGGER reorder

   AFTER UPDATE OF parts_on_hand ON inventory

   WHEN (new.parts_on_hand < new.reorder_point)

   FOR EACH ROW
   DECLARE
      NUMBER X;
   BEGIN
      SELECT COUNT(*) INTO X
      FROM pending_orders
      WHERE part_no = :new.part_no;
      IF X=0
      THEN
         INSERT INTO pending_orders
         VALUES (new.part_no, new.reorder_quantity, sysdate);
      END IF;
   END;
```

When the triggering event is an UPDATE statement, you can include a column list to identify which columns must be updated to fire the trigger.

You cannot specify a column list for INSERT and DELETE statements, because they affect entire rows of information.

Row and Statement Triggers/ Before and After

- For a single table you can create 3 of each type, one for each of the commands DELETE, INSERT and UPDATE making 12 triggers. (There is also an INSTEAD_OF trigger)
- You can also create triggers that fire for more than one command

		For Each Row option
BEFORE option	*BEFORE statement trigger*: Oracle fires the trigger once before executing the triggering statement	*BEFORE row trigger*: Oracle fires the trigger before modifying each row affected by the triggering statement
AFTER option	*AFTER statement trigger*: Oracle fires the trigger once after executing the triggering statement	*AFTER row trigger*: Oracle fires the trigger after modifying each row affected by the triggering statement

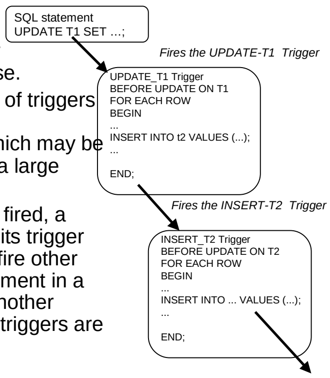
Multiple triggers

- Multiple triggers of the same type for the same statement for any given table.
  - two BEFORE statement triggers for UPDATE statements on the ENROL table.
- Multiple types of DML statements can fire a trigger,
  - can use conditional predicates to detect the type of triggering statement, hence
  - can create a single trigger that executes different code based on the type of statement that fires the trigger.

```
CREATE TRIGGER at AFTER UPDATE OR DELETE OR INSERT ON student
DECLARE  typeofupdate  CHAR(8); BEGIN
IF updating THEN typeofupdate := 'update'; …..END IF;
IF deleting  THEN typeofupdate := 'delete'; ……END IF;
IF inserting THEN typeofupdate := 'insert';  ……END IF;
…..
```

Some Cautionary Notes about Triggers

- Triggers are useful for customizing a database.
- But the excessive use of triggers can result in complex interdependencies, which may be difficult to maintain in a large application.
- E.g., when a trigger is fired, a SQL statement within its trigger action potentially can fire other triggers. When a statement in a trigger body causes another trigger to be fired, the triggers are said to be *cascading*.



The Execution Model for Triggers

- A single SQL statement can potentially fire up to four types of triggers: BEFORE row triggers, BEFORE statement triggers, AFTER row triggers, and AFTER statement triggers.
- A triggering statement or a statement within a trigger can cause one or more integrity constraints to be checked.
- Triggers can contain statements that cause other triggers to fire (cascading triggers).
- Oracle uses an execution model to maintain the proper firing sequence of multiple triggers and constraint checking

How Triggers Are Used

- Could restrict DML operations against a table to those issued during regular business hours.
- Could restrict DML operations to occur only at certain times during weekdays.
- Other uses:
  - automatically generate derived column values
  - prevent invalid transactions
  - enforce referential integrity across nodes in a distributed database
  - provide transparent event logging
  - provide sophisticated auditing
  - maintain synchronous table replicates
  - gather statistics on table access

Triggers vs. Declarative Integrity Constraints

- Triggers allow you to *define* and *enforce* integrity rules, but is not the same as an integrity constraint.
- A trigger defined to enforce an integrity rule does not check data already loaded into a table.
- You use database triggers only
  - when a required referential integrity rule cannot be enforced using the following integrity constraints: NOT NULL, UNIQUE key, PRIMARY KEY, FOREIGN KEY, CHECK, update CASCADE, update and delete SET NULL, update and delete SET DEFAULT
  - to enforce referential integrity when child and parent tables are on different nodes of a distributed database
  - to enforce complex business rules not definable using integrity constraints

Modifying Views

- Modifying views has inherent problems of ambiguity.
  - Deleting a row in a view could either mean
    - deleting it from the base table or
    - updating some column values so that it will no longer be selected by the view.
  - Inserting a row in a view could either mean
    - inserting a new row into the base table or
    - updating an existing row so that it will be projected by the view.
  - Updating a column in a view that involves joins might change the semantics of other columns that are not projected by the view.

Triggers and Views

- Triggers can be defined only on tables, not on views but triggers on the base table(s) of a view are fired if an INSERT, UPDATE, or DELETE statement is issued against a view.
- INSTEAD OF triggers provide a transparent way of modifying views that cannot be modified directly through SQL DML statements (INSERT, UPDATE, and DELETE).
- Oracle fires the INSTEAD OF trigger instead of executing the triggering statement. The trigger performs update, insert, or delete operations directly on the underlying tables.
- Users write normal INSERT, DELETE, and UPDATE statements against the view and the INSTEAD OF trigger works invisibly in the background to make the right actions take place.
- By default, INSTEAD OF triggers are activated for each row.

```
CREATE VIEW tutor_info AS
SELECT s.name,s.studno,s.tutor,t.roomno
FROM   student s, staff t
WHERE  s.tutor = t.lecturer;
```

Additional material

The Execution Model for Triggers

1. Execute all BEFORE statement triggers that apply to the statement.
2. Loop for each row affected by the SQL statement.
   a. Execute all BEFORE row triggers that apply to the statement.
   b. Lock and change row, and perform integrity constraint checking. (The lock is not released until the transaction is committed.)
   c. Execute all AFTER row triggers that apply to the statement.
3. Complete deferred integrity constraint checking.
4. Execute all AFTER statement triggers that apply to the statement.

Example of an INSTEAD OF Trigger

```
CREATE TRIGGER tutor_info_insert
INSTEAD OF INSERT ON tutor_info
REFERENCING NEW AS n -- new tutor
FOR EACH ROW
```

The actions shown for rows being inserted into the TUTOR_INFO view first test to see if appropriate rows already exist in the base tables from which TUTOR_INFO is derived. The actions then insert new rows or update existing rows, as appropriate. Similar triggers can specify appropriate actions for UPDATE and DELETE.

```
BEGIN
IF NOT EXISTS SELECT * FROM student WHERE student.studno = :n.studno
THEN  INSERT INTO student(studentno,name,tutor)
               VALUES(:n.studno, :n.name, :n.tutor);
ELSE  UPDATE student SET student.tutor = :n.tutor
               WHERE student.studno = :n.studno;
END IF;
IF NOT EXISTS SELECT * FROM staff WHERE staff.lecturer = :n.tutor
THEN  INSERT INTO staff VALUES(:n. staff.tutor, :n.roomno);
ELSE UPDATE staff SET staff.roomno = :n.roomno WHERE staff.lecturer =
   :n.tutor;
END IF;
END;
```