Database Design
Referential Integrity
Constraint

Sunnie Chung
Referential Integrity Constraints

Idea: prevent “dangling tuples” (e.g.: a loan with a bname, Kenmore, when no Kenmore tuple in branch)

Referential Integrity:
ensure that:
foreign key value $\rightarrow$ primary key value

(note: need not to ensure $\leftarrow$, i.e., not all branches have to have loans)
Referential Integrity Constraints

Referencing Relation (e.g. loan)

Referenced Relation (e.g. branch)

In SQL:

```
CREATE TABLE branch(
    bname CHAR(15) PRIMARY KEY
    ....)
```

```
CREATE TABLE loan ( 
    .......
    bname
    FOREIGN KEY bname REFERENCES branch);
```

Affects:
1) Insertions, updates of referencing relation (FK changes)
2) Deletions, updates of referenced relation (PK changes)
Two Types of Problems in Maintaining Referential Integrity

Problem 1:

Insert/Update with Invalid FK Value

- Non existing PK value or
- Any symbols other than Null

Solution by Server:

*Check if the FK value is an existing PK value or Null whenever it is submitted to be inserted/updated*
Two Types of Problems in Maintaining Referential Integrity

Problem 2:

When Delete/Insert/Update PK Value Fail to Propagate Deleted/Inserted/Updated PK Value to All the FKS that are pointing to PK

Solutions:

Add CASCADE in DDL to create Automatic System Trigger to Propagate the PK Change to ALL FKS

Create (Implement) Triggers to Propagate the PK Change to ALL FKS
Enforcing Referential Integrity

Consider Students\(\text{sid}, \text{name}, \text{gpa}\) and Enrolled \(\text{rid}, \text{semester}, \text{sid}\);
\text{sid} in Enrolled is a foreign key that references Students.

What should be done if an Enrolled tuple with a non-existent student id is inserted? **Reject it!**

What should be done if a Students tuple is deleted?

- **Also delete all Enrolled tuples that refer to it.**
- **Disallow deletion of a Students tuple that is referred to.**
- **Set sid in Enrolled tuples that refer to it to a default sid.**
  *(In SQL, also: Set sid in Enrolled tuples that refer to it to a special value null, denoting `unknown’ or `inapplicable’).*

**Similar if primary key of Students tuple is updated.**
Referential Integrity Constraints

Referential Integrity Constraints

what happens when we try to delete this tuple?

Ans: 3 possibilities

1) reject deletion/ update

2) set $t_i[c], t_j[c] = NULL$

3) propagate deletion/update
   DELETE: delete $t_i, t_j$
   UPDATE: set $t_i[c], t_j[c]$ to updated values
Referential Integrity Constraints

CREATE TABLE A ... FOREIGN KEY c REFERENCES B 

Action: 1) left blank (deletion/update rejected)

2) ON DELETE SET NULL/ ON UPDATE SET NULL
   sets ti[c] = NULL, tj[c] = NULL

3) ON DELETE CASCADE
   deletes ti, tj
   ON UPDATE CASCADE
   sets ti[c], tj[c] to new key values
Global Constraints

Idea: two kinds

1) single relation (constraints spans multiple columns)

   E.g.: CHECK (total = svngs + check) declared in the CREATE TABLE

2) multiple relations: CREATE ASSERTION

SQL examples:

  1) single relation: All Bkln branches must have assets > 5M

     CREATE TABLE branch (  
           ........
           bcity CHAR(15),
           assets INT,
           CHECK (NOT(bcity = 'Bkln') OR assets > 5M))

   Affects:
           insertions into branch
           updates of bcity or assets in branch
Global Constraints

SQL example:
2) Multiple relations: every loan has a borrower with a savings account

CHECK (NOT EXISTS (=
    SELECT  *
    FROM loan AS L
    WHERE NOT EXISTS (=
        SELECT  *
        FROM borrower B, depositor D, account A
        WHERE B.cname = D.cname AND
        D.acct_no = A.acct_no AND
        L.lno = B.lno)))

Problem: Where to put this constraint? At depositor? Loan? ....

Ans: None of the above:
CREATE ASSERTION loan-constraint
    CHECK ( ..... )
Checked with EVERY DB update!
very expensive.....
## Summary: Integrity Constraints

<table>
<thead>
<tr>
<th>Constraint Type</th>
<th>Where declared</th>
<th>Affects...</th>
<th>Expense</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Constraints</td>
<td>CREATE TABLE (PRIMARY KEY, UNIQUE)</td>
<td>Insertions, Updates</td>
<td>Moderate</td>
</tr>
<tr>
<td>Attribute Constraints</td>
<td>CREATE TABLE CREATE DOMAIN (Not NULL, CHECK)</td>
<td>Insertions, Updates</td>
<td>Cheap</td>
</tr>
<tr>
<td>Referential Integrity</td>
<td>Table Tag (FOREIGN KEY .... REFERENCES ....)</td>
<td>1. Insertions into referencing rel’n</td>
<td>1,2: like key constraints. Another reason to index/sort on the primary keys</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Updates of referencing rel’n of relevant attrs</td>
<td>3,4: depends on</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. Deletions from referenced rel’n</td>
<td>a. update/delete policy chosen</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4. Update of referenced rel’n</td>
<td>b. existence of indexes on foreign key</td>
</tr>
<tr>
<td>Global Constraints</td>
<td>Table Tag (CHECK)</td>
<td>1. For single rel’n constraint, with insertion, deletion of relevant attrs</td>
<td>1. cheap</td>
</tr>
<tr>
<td></td>
<td>or outside table (CREATE ASSERTION)</td>
<td>2. For assertions w/ every db modification</td>
<td>2. very expensive</td>
</tr>
</tbody>
</table>