Semantic Constraints (cont’d)

• Used for application dependent conditions
• Example: limit attribute values

CREATE TABLE Transcript ( 
    StudId: INTEGER, 
    CrsCode: CHAR(6), 
    Semester: CHAR(6), 
    Grade: CHAR(1), 
    CHECK (Grade IN ('A', 'B', 'C', 'D', 'F')), 
    CHECK (StudId > 0 AND StudId < 1000000000) 
)

• Each row in table must satisfy condition

Semantic Constraints (cont’d)

• Example: relate values of attributes in different columns

CREATE TABLE Employee ( 
    Id: INTEGER, 
    Name: CHAR(20), 
    Salary: INTEGER, 
    MngrSalary: INTEGER, 
    CHECK (MngrSalary > Salary) 
)

Constraints – Problems

• Problem 1: Empty table always satisfies all CHECK constraints (an idiosyncrasy of the SQL standard)

CREATE TABLE Employee ( 
    Id: INTEGER, 
    Name: CHAR(20), 
    Salary: INTEGER, 
    MngrSalary: INTEGER, 
    CHECK (0 < (SELECT COUNT(*) FROM Employee)) 
)

– If Employee is empty, there are no rows on which to evaluate the CHECK condition.

Constraints – Problems

• Problem 2: Inter-relational constraints should be symmetric

CREATE TABLE Employee ( 
    Id: INTEGER, 
    Name: CHAR(20), 
    Salary: INTEGER, 
    MngrSalary: INTEGER, 
    CHECK ((SELECT COUNT(*) FROM Employee) < (SELECT COUNT(*) FROM Manager)) 
)

– Why should constraint be in Employee and not Manager?
– What if Employee is empty?

Assertion

• Element of schema (like table)
• Symmetrically specifies an inter-relational constraint
• Applies to entire database (not just the individual rows of a single table)
  – hence it works even if Employee is empty

CREATE ASSERTION Don’tFireEveryone 
CHECK (0 < SELECT COUNT(*) FROM Employee)

Assertion

CREATE ASSERTION KeepEmployeeSalariesDown 
CHECK (NOT EXISTS( 
    SELECT * FROM Employee E 
    WHERE E.Salary > E.MngrSalary))
**Assertions and Inclusion Dependency**

CREATE ASSERTION NoEmptyCourses
CHECK (NOT EXISTS (SELECT * FROM Teaching T
WHERE -- for each row T check
-- the following condition
NOT EXISTS (SELECT * FROM Transcript R
WHERE T.CrsCode = R.CrsCode
AND T.Semester = R.Semester))
)

Courses with no students

Students in a particular course

---

**Domains**

- Possible attribute values can be specified
  - Using a CHECK constraint or
  - Creating a new domain
- Domain can be used in several declarations
- Domain is a schema element

CREATE DOMAIN Grades CHAR (1)
CHECK (VALUE IN ('A', 'B', 'C', 'D', 'F'))

CREATE TABLE Transcript
(…,
Grade: Grades Grades,
… )

---

**Foreign Key Constraint**

CREATE TABLE Teaching
(
ProfIId: INTEGER,
CrsCode: CHAR (6),
Semester: CHAR (6),
PRIMARY KEY (CrsCode, Semester),
FOREIGN KEY (CrsCode) REFERENCES Course,
FOREIGN KEY (ProfIId) REFERENCES Professor (Id )
)

---

**Circularity in Foreign Key Constraint**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>y</td>
</tr>
</tbody>
</table>

Problem 1: Creation of A requires existence of B and vice versa
Solution: CREATE TABLE A (...) -- no foreign key
CREATE TABLE B (...) -- include foreign key
ALTER TABLE A
ADD CONSTRAINT cons
FOREIGN KEY (A) REFERENCES B (B )

---

**Circularity in Foreign Key Constraint (cont’d)**

- Problem 2: Insertion of row in A requires prior existence of row in B and vice versa
- Solution: use appropriate constraint checking mode:
  - IMMEDIATE checking
  - DEFERRED checking
Reactive Constraints

- Constraints enable DBMS to recognize a bad state and reject the statement or transaction that creates it.
- More generally, it would be nice to have a mechanism that allows a user to specify how to react to a violation of a constraint.
- SQL-92 provides a limited form of such a reactive mechanism for foreign key violations.

Handling Foreign Key Violations

- Insertion into A: Reject if no row exists in B containing foreign key of inserted row.
- Deletion from B:
  - **NO ACTION**: Reject if row(s) in A references row to be deleted (default response).
  - **SET NULL**: Set value of foreign key in referencing row(s) in A to null.
  - **SET DEFAULT**: Set value of foreign key to default value (y) which must exist in B.
  - **CASCADE**: Delete referencing row(s) in A as well.

Handling Foreign Key Violations (cont’d)

- Deletion from B (cont’d):
  - **SET NULL**: Set value of foreign key in referencing row(s) in A to null.

Handling Foreign Key Violations (cont’d)

- Deletion from B (cont’d):
  - **SET DEFAULT**: Set value of foreign key in referencing row(s) in A to default value (y) which must exist in B.

Handling Foreign Key Violations (cont’d)

- Update (change) foreign key in A: Reject if no row exists in B containing new foreign key.
- Update candidate key in B (to z) – same actions as with deletion:
  - **NO ACTION**: Reject if row(s) in A references row to be updated (default response).
  - **SET NULL**: Set value of foreign key to null.
  - **SET DEFAULT**: Set value of foreign key to default.
  - **CASCADE**: Propagate z to foreign key.
Handling Foreign Key Violations (cont’d)

- The action taken to repair the violation of a foreign key constraint in A may cause a violation of a foreign key constraint in C
  - The action specified in C controls how that violation is handled;
  - If the entire chain of violations cannot be resolved, the initial deletion from B is rejected.

```
+---+---+---+
| A | B | C |
+---+---+---+
| x | y | y |
```

Specifying Actions

```sql
CREATE TABLE Teaching (  
    ProfId INTEGER,  
    CrsCode CHAR(6),  
    Semester CHAR(6),  
    PRIMARY KEY (CrsCode, Semester),  
    FOREIGN KEY (ProfId) REFERENCES Professor (Id)  
        ON DELETE NO ACTION  
        ON UPDATE CASCADE,  
    FOREIGN KEY (CrsCode) REFERENCES Course (CrsCode)  
        ON DELETE SET NULL  
        ON UPDATE CASCADE )
```

Triggers

- A more general mechanism for handling events
  - Not in SQL-92, but is in SQL:1999
- Trigger is a schema element (like table, assertion, …)

```sql
CREATE TRIGGER CrsChange  
AFTER UPDATE OF CrsCode, Semester ON Transcript  
WHEN (Grade IS NOT NULL)  
ROLLBACK
```

Views

- Schema element
- Part of external schema
- A virtual table constructed from actual tables on the fly
  - Can be accessed in queries like any other table
  - Not materialized, constructed when accessed
  - Similar to a subroutine in ordinary programming

```
CREATE VIEW CoursesTaken (StudId, CrsCode, Semester) AS  
SELECT T.StudId, T.CrsCode, T.Semester  
FROM Transcript T

CREATE VIEW CoursesITook (CrsCode, Semester, Grade) AS  
SELECT T.CrsCode, T.Semester, T.Grade  
FROM Transcript T  
WHERE T.StudId = '123456789'
```

Modifying the Schema

```sql
ALTER TABLE Student  
ADD COLUMN Gpa INTEGER DEFAULT 0

ALTER TABLE Student  
ADD CONSTRAINT GpaRange  
CHECK (Gpa >= 0 AND Gpa <= 4)

ALTER TABLE Transcript  
DROP CONSTRAINT Cons -- constraint names are useful

DROP TABLE Employee

DROP ASSERTION Don’tFireEveryone
```
Access Control

• Databases might contain sensitive information
• Access has to be limited:
  – Users have to be identified – authentication
  • Generally done with passwords
  – Each user must be limited to modes of access appropriate to that user - authorization
• SQL-92 provides tools for specifying an authorization policy but does not support authentication (vendor specific)

Controlling Authorization in SQL

<table>
<thead>
<tr>
<th>GRANT access_list</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON table</td>
</tr>
<tr>
<td>TO user_list</td>
</tr>
</tbody>
</table>

access modes: SELECT, INSERT, DELETE, UPDATE, REFERENCES

GRANT UPDATE (Grade) ON Transcript TO prof_smith
  – Only the Grade column can be updated by prof_smith

GRANT SELECT ON Transcript TO joe
  – Individual columns cannot be specified for SELECT access (in the SQL standard) – all columns of Transcript can be read
  – But SELECT access control to individual columns can be simulated through views (next)

Authorization Mode REFERENCES

• Foreign key constraint enforces relationship between tables that can be exploited to
  – Control access: can enable perpetrator prevent deletion of rows
  
  CREATE TABLE DontDismissMe
  (Id INTEGER,
   FOREIGN KEY (Id) REFERENCES Student
   ON DELETE NO ACTION)
  
  – Reveal information: successful insertion into
  DontDismissMe means a row with foreign key value exists in Student
  
  INSERT INTO DontDismissMe ("11111111")

REFERENCE Access mode (cont’d)

<table>
<thead>
<tr>
<th>GRANT REFERENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON Student</td>
</tr>
<tr>
<td>TO joe</td>
</tr>
</tbody>
</table>