Lecture 03 (Chapter 03)
SQL

Lecturer: Sael Lee

Slide adapted from the author’s slides and Dr. Ilchul Yoon’s slides.
SQL

- Language for describing database schema & operations on tables

- Data Definition Language (DDL): sublanguage of SQL for describing schema
Tables

- SQL entity that corresponds to a relation
- An element of the database schema
Table Declaration

CREATE TABLE Student (
    Id: INTEGER,
    Name: CHAR(20),
    Address: CHAR(50),
    Status: CHAR(10)
)

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Address</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>101222333 234567890</td>
<td>John  Mary</td>
<td>10 Cedar St 22 Main St</td>
<td>Freshman Sophomore</td>
</tr>
</tbody>
</table>
CREATE TABLE Course (  
  CrsCode CHAR(6),  
  CrsName CHAR(20),  
  DeptId CHAR(4),  
  Descr CHAR(100),  
  PRIMARY KEY (CrsCode),  
  UNIQUE (DeptId, CrsName)  
) -- candidate key

Comments start with 2 dashes
Null

- Problem: Not all information might be known when row is inserted (e.g., *Grade* might be missing from Transcript)
- A column might not be applicable for a particular row (e.g., MaidenName if row describes a male)

- Solution: Use place holder – null
  - Not a value of any domain (although called null value)
    - Indicates the absence of a value
  - Not allowed in certain situations
    - Primary keys and columns constrained by NOT NULL
Default Value

- Value to be assigned if attribute value in a row is not specified

```sql
CREATE TABLE Student(
    Id INTEGER,
    Name CHAR(20) NOT NULL,
    Address CHAR(50),
    Status CHAR(10) DEFAULT 'freshman',
    PRIMARY KEY (Id)
)
```
Semantic Constraints in SQL

- Primary key and foreign key are examples of structural constraints
- Semantic constraints
  - Express the logic of the application at hand:
    - e.g., number of registered students $\leq$ maximum enrollment
Semantic Constraints (cont’d)

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

```sql
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

```sql
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)

```sql
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

```sql
CREATE TABLE Employee (  
  Id   INTEGER,  
  Name CHAR(20),  
  Salary   INTEGER,  
  MgrSalary  INTEGER,  
  CHECK ((SELECT COUNT(*) FROM Manager) < (SELECT COUNT(*) FROM Employee))  
)
```

- 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)
  - Does it work even if Employee is empty?

```
CREATE ASSERTION DontFireEveryone
CHECK (0 < SELECT COUNT(*) FROM Employee)
```
Assertion

```sql
CREATE ASSERTION KeepEmployeeSalariesDown
CHECK (NOT EXISTS(
    SELECT * FROM Employee E
    WHERE E.Salary > E.MngrSalary))
```
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)
  ))

Referential integrity constraint that is not a foreign key constraint
(CrsCode, Semester) of Teaching references
(CrsCode, Semester) of Transcript

Target attributes is not a CK in Transcript
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,
    ... )
```
CREATE TABLE Teaching ( 
    ProfId  INTEGER,  
    CrsCode CHAR(6),  
    Semester CHAR(6), 
    PRIMARY KEY (CrsCode, Semester), 
    FOREIGN KEY (CrsCode) REFERENCES Course, 
    FOREIGN KEY (ProfId) REFERENCES Professor (Id) )
Foreign Key Constraint

CrsCode ProfId

CrsCode

Course

Teaching

Professor

Id

x

y
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_3) REFERENCES B (B_1)
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)

Request to delete row rejected
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

```
A          B
null
```

*Row deleted*
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)

- Deletion from B (cont’d):
  - **CASCADE**: Delete referencing row(s) in A as well

```
A B
x x
```
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

Cascading when key in B changed from x to z
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.
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

Guard
```
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
Views - Examples

- Part of external schema suitable for use in Bursar’s office:

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

- Part of external schema suitable for student with Id 123456789:

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

\[\text{ALTER TABLE Student} \]
\[\text{ADD COLUMN Gpa INTEGER DEFAULT } 0\]

\[\text{ALTER TABLE Student} \]
\[\text{ADD CONSTRAINT GpaRange CHECK (Gpa } \geq 0 \text{ AND } Gpa \leq 4)\]

\[\text{ALTER TABLE Transcript} \]
\[\text{DROP CONSTRAINT Cons} \]
\[-- \text{constraint names are useful}\]

\[\text{DROP TABLE Employee}\]

\[\text{DROP ASSERTION DontFireEveryone}\]
Constraint Name Example

CREATE TABLE TRANSCRIPT (  
    StudID INTEGER,
    CrsCode CHAR(6),
    Semester CHAR(6),
    Grade GRADES,
    CONSTRAINT TRKEY PK (Sid, C, Sem)
    CONSTRAINT STUDFK FK (Sid) REFERENCES STUDENT,
    CONSTRAINT CRSFK FK (C) REFERENCES COURSE,
    CONSTRAINT IDRANGE CHECK ( Sid > 0 AND Sid < 100000) )

ALTER TABLE TRANSCRIPT DROP CONSTRAINT STUDFK
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

```sql
GRANT access_list
   ON table
TO user_list    [WITH GRANT OPTION]
```

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

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

```sql
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)
```
Controlling Authorization in SQL Using Views

- GRANT SELECT ON CoursesTaken TO joe

  \[\text{GRANT access ON view TO user\_list}\]

- Thus views can be used to simulate access control to individual columns of a table
Authorization Mode REFERENCES

- Foreign key constraint enforces relationship between tables that can be exploited to
  - Control access: can enable perpetrator prevent deletion of rows
  - Reveal information: successful insertion into DontDissmissMe means a row with foreign key value exists in Student

```
CREATE TABLE DontDismissMe (
  Id INTEGER,
  FOREIGN KEY (Id) REFERENCES Student
  ON DELETE NO ACTION
)
```

![GRANT REFERENCES ON Student TO Joe](image)

```
INSERT INTO DontDismissMe ('111111111')
```