Relational Data Model

Chapter 2.1, 2.2 & Chapter 3

Topics

• Relational Databases
  – Tables & SQL (DML)
• Data Independence
• Relational Data Model
• Integrity Constraints
• SQL (DDL) and Relational Data Model

Data Model

• Underlying the structure of a database is a **Data Model**
  – A collection of conceptual tools for describing:
    • Data
    • Data relationships
    • Data semantics
    • Consistency constraints
• Different data models have been proposed over the past 35-40 years
  • Network (legacy)
  • Hierarchical (legacy)
  • Object-oriented (niche)

Relational Databases

• Majority of applications being used and developed today use relational databases
• Built on the **Relational Data Model**
• Conceptually data is stored in tables

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Address</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>John</td>
<td>123 Main</td>
<td>fresh</td>
</tr>
<tr>
<td>2222</td>
<td>Mary</td>
<td>321 Oak</td>
<td>soph</td>
</tr>
<tr>
<td>1234</td>
<td>Bob</td>
<td>444 Pine</td>
<td>soph</td>
</tr>
<tr>
<td>9999</td>
<td>Joan</td>
<td>777 Grand</td>
<td>senior</td>
</tr>
</tbody>
</table>
Table

- Each table has a set of rows (no duplicates)
- Each row describes a different entity
- Each column states a particular fact about each entity
- Each column has an associated domain from which it gets its values
  - Domain of Status = {fresh, soph, junior, senior}

Manipulating Data: Structured Query Language (SQL)

- Language for manipulating tables
- **Declarative** –
  - Statement specifies what needs to be obtained,
  - *not how* it is to be achieved (e.g., how to access data, the order of operations)
- Due to declarative nature of SQL, DBMS determines evaluation strategy
  - This greatly simplifies application programs
  - But DBMS is not infallible:
    - programmers should have an idea of strategies used by DBMS so they can design better tables, indexes, statements, in such a way that DBMS can evaluate statements efficiently

Structured Query Language (SQL)

- Language for constructing a new table from argument table(s).
  - FROM indicates source tables
  - WHERE indicates which rows to retain
    - It acts as a filter
  - SELECT indicates which columns to extract from retained rows
    - Projection
  - The result is a table

Example

\[
\text{SELECT Name FROM Student WHERE } \text{Id} > 4999
\]

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Address</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>John</td>
<td>123 Main</td>
<td>fresh</td>
</tr>
<tr>
<td>5522</td>
<td>Mary</td>
<td>77 Pine</td>
<td>senior</td>
</tr>
<tr>
<td>9876</td>
<td>Bill</td>
<td>83 Oak</td>
<td>junior</td>
</tr>
</tbody>
</table>
Examples

SELECT Id, Name FROM Student

SELECT Id, Name FROM Student WHERE Status = 'senior'

SELECT * FROM Student WHERE Status = 'senior'

SELECT COUNT(*) FROM Student WHERE Status = 'senior'

More Complex Example

• Goal: table in which each row names a senior and gives a course taken and grade
• Combines information in two tables:
  – Student: Id, Name, Address, Status
  – Transcript: StudId, CrsCode, Semester, Grade

SELECT Name, CrsCode, Grade
FROM Student, Transcript
WHERE StudId = Id AND Status = 'senior'

Join

SELECT a1, b1
FROM T1, T2
WHERE a2 = b2

FROM T1, T2
yields:

WHERE a2 = b2
yields:

SELECT a1, b1
yields result:

Modifying Table Data

UPDATE Student
SET Status = 'soph'
WHERE Id = 111111111

INSERT INTO Student (Id, Name, Address, Status)
VALUES (999999999, 'Bill', '432 Pine', 'senior')

DELETE FROM Student
WHERE Id = 111111111
Topics

- Relational Databases
  - Tables & SQL (DML)
- Data Independence
- Relational Data Model
- Integrity Constraints
- SQL (DDL) and Relational Data Model

Data and Its Structure

- Data is actually stored as bits, but it is difficult to work with data at this level.
- It is convenient to view data at different *levels of abstraction*.
- **Schema**: Description of data at some abstraction level. Each level has its own schema.
- We will be concerned with three schemas: *physical*, *conceptual*, and *external*.

Physical Data Level

- **Physical schema** describes details of how data is stored: tracks, cylinders, indices etc.
- Early applications worked at this level – explicitly dealt with details
- **Problem**: Routines were hard-coded to deal with physical representation
  - Changes to data structure difficult to make
  - Application code becomes complex since it must deal with details
  - Rapid implementation of new features impossible

Conceptual Data Level

- Hides details.
  - In the relational model, the conceptual schema presents data as a set of tables
- DBMS maps from conceptual to physical schema automatically
- Physical schema can be changed without changing application:
  - DBMS would change mapping from conceptual to physical transparently
  - This property is referred to as *physical data independence*
External Data Level

- In the relational model, the *external schema* also presents data as a set of relations
- An external schema specifies a *view* of the data in terms of the conceptual level. It is tailored to the needs of a particular category of users
  - Portions of stored data should not be seen by some users
    - Students should not see their files in full
    - Faculty should not see billing data
  - Information that can be derived from stored data might be viewed as if it were stored
    - GPA not stored, but calculated when needed

External Data Level (contd)

- Application is written in terms of an external schema
- A view is computed when accessed (not stored)
- Different external schemas can be provided to different categories of users
- Translation from external to conceptual done automatically by DBMS at run time
- Conceptual schema can be changed without changing application:
  - Mapping from external to conceptual must be changed
- Referred to as *conceptual data independence*

Levels of Abstraction

- Payroll
- Billing
- Records

View 1 → View 2 → View 3

Conceptual schema → Physical schema
### Relational Database: Definitions

1. **Tables & SQL (DML)**

2. **Data Independence**
3. **Relational Data Model**
4. **Integrity Constraints**
5. **SQL (DDL) and Relational Data Model**

---

**Relational Database: Definitions (contd.)**

2. **Instance**: a set of tuples
   - All tuples have the same set of named attributes
   - Value of an attribute is drawn from the attribute’s domain

3. **Mathematical term ~ Database term**
   - Relation ~ Table
   - Tuple ~ Row
   - Attribute ~ Column

---

**Relation Instance (Example)**

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Address</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111111</td>
<td>John</td>
<td>123 Main</td>
<td>freshman</td>
</tr>
<tr>
<td>2345678</td>
<td>Mary</td>
<td>456 Cedar</td>
<td>sophmore</td>
</tr>
<tr>
<td>4433322</td>
<td>Art</td>
<td>77 So. 3rd</td>
<td>senior</td>
</tr>
<tr>
<td>7654321</td>
<td>Pat</td>
<td>88 No. 4th</td>
<td>sophmore</td>
</tr>
</tbody>
</table>

**Student**
Observations & More Definitions

• Relation is a set of tuples
  – Tuple ordering immaterial
  – No duplicates
• Attributes of a tuple are named
  – ordering of attributes is immaterial
• In addition to a value from an attribute’s domain
  – There is also a special value null (value unknown or undefined), which belongs to no domain
• Cardinality of relation = number of tuples
• Arity of relation = number of attributes

Relation Schema

• Relation name
• Attribute name & domains
• Integrity constraints like
  – The values of a particular attribute in all tuples are unique
  – The values of a particular attribute in all tuples are greater than 0
• Default values

Relational Database

• Finite set of relations
• Each relation consists of a schema and an instance
• Database schema = set of relation schemas and constraints among relations (inter-relational constraints)
• Database instance = set of (corresponding) relation instances

Database Schema (Example)

• Student (Id: INT, Name: STRING, Address: STRING, Status: STRING)
• Professor (Id: INT, Name: STRING, DeptId: DEPTS)
• Course (DeptId: DEPTS, CrsName: STRING, CrsCode: COURSES)
• Transcript (CrsCode: COURSES, StudId: INT, Grade: GRADES, Semester: SEMESTERS)
• Department(DeptId: DEPTS, Name: STRING)
Operations

- Operations on relations are precisely defined
  - Take relation(s) as argument, produce new relation as result
  - Unary (e.g., delete certain rows)
  - Binary (e.g., union, Cartesian product)
- Mathematically based
  - Expressions (= queries) can be analyzed by DBMS
  - Queries are transformed to equivalent expressions automatically (query optimization)

\[ \text{op1}(T1, \text{op2}(T2)) = \text{op3}(\text{op2}(T1), T2) \]

Topics

- Relational Databases
  - Tables & SQL (DML)
- Data Independence
- Relational Data Model
- Integrity Constraints
- SQL (DDL) and Relational Data Model

Integrity Constraints

- Part of schema
- Restriction on state (or of sequence of states) of database
- Automatically enforced by DBMS
  - Protects database from errors
  - Enforces enterprise rules
- Intra-relational - involve only one relation
  - Part of relation schema
  - e.g., all ids are unique
- Inter-relational - involve several relations
  - Part of relation schema or database schema

Kinds of Integrity Constraints

- Static – restricts legal states of database
  - Syntactic (structural)
    - e.g., all values in a column must be unique
  - Semantic (involve meaning of attributes)
    - e.g., cannot register for more than 18 credits
- Dynamic – limitation on sequences of database states
  - e.g., cannot raise salary by more than 5%
Key Constraint

- A key constraint is a sequence of attributes \( A_1, \ldots, A_n \) (\( n = 1 \) possible) of a relation schema, \( S \), with the following properties:
  - A relation instance \( s \) of \( S \) satisfies the key constraint iff at most one row in \( s \) can contain a particular set of values, \( a_1, \ldots, a_n \), for the attributes \( A_1, \ldots, A_n \)
  - Minimality: no subset of \( A_j, \ldots, A_k \) is a key constraint

- Key
  - Set of attributes mentioned in a key constraint
    - e.g., \( Id \) in Student
    - e.g., \((StudId, CrsCode, Semester)\) in Transcript
  - It is minimal: no subset of a key is a key
    - \((Id, Name)\) is not a key of Student

Super Key Constraint

- Set of attributes containing key
  - Minimality is not a requirement

- Any set of attributes that contains the key is a superkey
  - So every key is also a super key
    - The converse is not true
  - \((Id, Name)\) is a superkey of Student

- Set of all attributes of a relation is a superkey

Key Constraint (cont’d)

- Every relation has at least one key

- A relation can have several keys
  - E.g., Course(CrsCode: STRING, DeptId: STRING, CrsName: STRING, Desc: STRING)
  - \{CrsCode\} is a key
  - \{DeptId, CrsName\} is a key

- If a relation has several keys, these are referred to as candidate keys

- One key though is often identified as primary key
  - Choice of primary key impacts performance (more later in the course)
  - (Probably) choose \{CrsCode\} as the primary key

Referential Integrity

- Item named in one relation must refer to tuples that describe that item in another

- Course (\( \text{DeptId: DEPTS, CrsName: STRING, CrsCode: COURSES} \))

- Transcript (\( \text{CrsCode: COURSES, StudId: INT, Grade: GRADES, Semester: SEMESTERS} \))
  - Transcript (\( \text{CrsCode} \)) references Course(\( \text{CrsCode} \))

- Professor (\( \text{Id: INT, Name: STRING, DeptId: DEPTS} \))

- Department(\( \text{DeptId: DEPTS, Name: STRING} \))
  - Professor(\( \text{DeptId} \)) references Department(\( \text{DeptId} \))
**Referential Integrity: Example**

- IMPORTANT: the tuples in R1 refer to the tuples in R2 by value, i.e., the value of attribute A2 in the referenced tuple in R2 is stored as attribute A1 in the referencing tuple in R1
- Reference is NOT by physical address (that would have meant using Physical Schema information at the Conceptual Schema level)

**Foreign Key Constraint**

- Attribute $A_1$ is a foreign key of R1 referring to attribute $A_2$ in R2, if whenever there is a value $v$ of $A_1$, there is a tuple of R2 in which $A_2$ has value $v$, and $A_2$ is a key of R2
  - This is a special case of referential integrity: $A_2$ must be a candidate key of R2
    - e.g., CrsCode is a key of Course in the above
    - If no row exists in R2 => violation of referential integrity

**Foreign Key Constraint (Example)**

- Foreign key might consist of several columns
  - $(\text{CrsCode}, \text{Semester})$ of Transcript references $(\text{CrsCode}, \text{Semester})$ of Teaching
- $R1(A_1, \ldots, A_n)$ references $R2(B_1, \ldots, B_m)$
  - $B_1, \ldots, B_m$ must be a candidate key of R2
  - $A_i$ and $B_i$ must have same domains
    - although not necessarily the same names
    - E.g.: Teaching$(\text{CrsCode}: \text{COURSES}, \text{Sem}: \text{SEMESTERS}, \text{ProfId}: \text{INT})$
  - Professor$(\text{Id}: \text{INT}, \text{Name}: \text{STRING}, \text{DeptId}: \text{DEPTS})$
  - $\text{ProfId}$ attribute of Teaching references $\text{Id}$ attribute of Professor
### Foreign Key (cont’d)

- Not all rows of R2 need to be referenced
  - E.g., some course might not be taught
- Value of a foreign key might be left specified
  - E.g., DeptId column of some professor might be **null**
- R1 and R2 need not be distinct.
  - Employee(Id:INT, MgrId:INT, ….)
    - Employee(MgrId) references Employee(Id)
  - Every manager is also an employee and hence has a unique row in Employee

### Inclusion Dependency

- Referential integrity constraint that is not a foreign key constraint
  - E.g., Transcript (CrsCode: COURSES, StudId: INT, Grade: GRADES, Semester: SEMESTERS)
  - Teaching(CrsCode: COURSES, Sem: SEMESTERS, ProfId: INT)
  - Teaching(CrsCode, Semester) references Transcript(CrsCode, Semester)
  - (no empty classes allowed)
  - Target attributes do not form a candidate key in Transcript
    - (StudId missing)

### 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 NULL Constraint

- If a set of attributes is designated as the primary key, then the value for these attributes cannot be **null**
- Otherwise by default an attribute value can be **null**
- **NOT NULL Constraint:**
  - Ensures that the value of a (non primary key) attribute will not be **null**
    - E.g., Professor(Id: INT, Name: STRING, DeptId: DEPTS)
  - A Professor cannot be added to the relation without knowing his/her name
  - Solution: Define a NOT NULL constraint on Name attribute
Topics

- Relational Databases
  - Tables & SQL (DML)
- Data Independence
- Relational Data Model
- Integrity Constraints
- SQL (DDL) and Relational Data Model

SQL & Relational Data Model

- SQL supports definition of:
  - Conceptual and external schema
    - Data definition language (DDL)
  - Integrity constraints, domains (DDL)
  - Operations on data
    - Data manipulation language (DML)
      - E.g., SELECT, UPDATE, INSERT, DELETE
  - Directives that influence the physical schema
    (affects performance, not semantics)
    - Storage definition language (SDL)

Tables

- SQL entity that corresponds to a relation
- An element of the database schema
- SQL-92 is currently the most supported standard but is now superseded by SQL:1999 and SQL:2003
- Database vendors generally deviate from the standard, but eventually converge

Table Declaration

```sql
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>10122233</td>
<td>John</td>
<td>10 Cedar St</td>
<td>Freshman</td>
</tr>
<tr>
<td>234567890</td>
<td>Mary</td>
<td>22 Main St</td>
<td>Sophomore</td>
</tr>
</tbody>
</table>

Student
System Catalog

- A DBMS needs information describing the structure of the database to process DML statements
  - E.g., For an INSERT statement, a unique key constraint would need to be checked
- This information is stored in the System Catalog
  - Conceptually: a set of special relations designed by the DBMS vendor and pre-created when a database is installed by the Database Administrator (DBA)
- When you execute a CREATE TABLE statement, a table gets created and certain rows describing this new table are automatically inserted in the System Catalog

System Catalog: Example

- If you look at SQL Plus Handout from last lecture, you will see the SQL query:
  - `select table_name from user_tables;`
- The relation `user_tables` is part of the System Catalog
- After creating a table, the above query will show that a row has been inserted in `user_tables` corresponding to this new table

Primary/Candidate Keys

CREATE TABLE Course (  
  CrsCode CHAR(6),  
  CrsName CHAR(20),  
  DeptId CHAR(4),  
  Descr CHAR(100),  
  PRIMARY KEY (CrsCode),  
  UNIQUE (DeptId, CrsName)  
)  

Null value not allowed for CrsCode

Comments start with 2 dashes

Default Value

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

  CREATE TABLE Student (  
    Id INTEGER,  
    Name CHAR(20) NOT NULL,  
    Address CHAR(50),  
    Status CHAR(10) DEFAULT 'freshman',  
    PRIMARY KEY (Id)  
  )

- Will need to use a somewhat different syntax for INSERT statement
  - INSERT INTO student (id, name) VALUES (1, 'John')
  - Inserted row will have values: 1, 'John', null, 'freshman'
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 ≤ maximum enrollment
  - Semantic constraints are generally defined using CHECK constraint mechanism

Semantic Constraints (cont’d)

- **Example**: An employee’s salary must be less than the manager’s salary
  
  ```sql
  CREATE TABLE Employee (
    Id INTEGER,
    Name CHAR(20),
    Salary INTEGER,
    MgrSalary INTEGER,
    CHECK ( MgrSalary > Salary )
  )
  ```

Inter-Relation Constraints

- E.g.: Number of employees must be more than number of managers
  
  ```sql
  CREATE TABLE Employee (
    Id INTEGER,
    Name CHAR(20),
    CHECK ( [SELECT COUNT(*) FROM Manager] < [SELECT COUNT(*) FROM Employee] )
  )
  ```

  - According to SQL2 such a constraint will be checked on every insertion/update to the table Employee, but not on insertion/updates to the table Manager
  - **Note**: Oracle does NOT allow a CHECK constraint to refer to another table

Assertion

- Element of schema (like table)
- Symmetrically specifies an inter-relational constraint
- Applies to entire database
  
  ```sql
  CREATE ASSERTION DontFireEveryone
  CHECK (0 < SELECT COUNT(*) FROM Employee)
  ```

  - **Note**: Oracle does NOT support assertions
Domains

- One way to restrict the domain for an attribute, is to use CHECK constraint to constrain the allowable value for an attribute.

```
CREATE TABLE transcript {
    StudentId integer,
    CrsCode char(6),
    ...
    Grade char(1),
    CHECK (Grade in ('A', 'B', 'C', 'D', 'F'))
}
```

Domains (contd)

- Alternative to CHECK constraint in table definition
  - Create a new domain
- Domain is a schema element
  - can be used in several declarations
```
CREATE DOMAIN Grades CHAR (1)
CHECK (VALUE IN ('A', 'B', 'C', 'D', 'F'))

CREATE TABLE Transcript {
    ...
    Grade : Grades,
    ...
}
```

- Note: Oracle does NOT support CREATE DOMAIN

Foreign Key Constraint

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

[Diagram of foreign key constraint relationships between Teaching, Course, and Professor tables]

[Diagram explaining the relationships and keys involved in the foreign key constraint]
Circularity in Foreign Key Constraint

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 (`A3`) REFERENCES `B` (`B1`)

Circularity in FK Constraint: Instance Level

- The solution described on the previous slide solves the circularity problem at the schema level
- But what about the instance level?
- Problem: Insertion of row in A requires prior existence of row in B and vice versa
- **Solution 1:**
  - Add row to Table A with a NULL value for the foreign key
  - After row that needs to be referenced has been added to Table B, update the row in Table A and set the correct value for the foreign key

Circularity in FK Constraint at Instance Level: Solution 2

- Utilizes the concept of **transactions**
  - Transactions bundle together related updates/inserts/deletes
  - Either all of these operations succeed or none succeeds
  - COMMIT statement ends the transaction, makes changes permanent
  - During a transaction (i.e., before a COMMIT is issued), it is ok for database to be in an inconsistent state
- Define constraint with appropriate **constraint checking mode**:
  - **IMMEDIATE** checking: constraint checked after every SQL statement
  - **DEFERRED** checking: constraint will be checked when transaction is being committed
- If constraints defined in DEFERRED checking mode
  - Insertion of a row to Table A allowed even when the referenced row has not been added to Table B
  - However, at the time you issue COMMIT the constraint must hold
  - Otherwise all your changes in the transaction are rejected!

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: Choice 1
  – NO ACTION: Reject if row(s) in A references row to be deleted (default response)

A B

Request to delete row rejected

• Deletion from B: Choice 2
  – SET NULL: Set value of foreign key in referencing row(s) in A to null

A B

Row deleted

• Deletion from B: Choice 3
  – SET DEFAULT: Set value of foreign key in referencing row(s) in A to default value (y) which must exist in B
  – Not supported in Oracle

A B

Row deleted

• Deletion from B: Choice 4
  – CASCADE: Delete referencing row(s) in A as well

A 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; the only one supported by Oracle)
  - SET NULL: Set value of foreign key to null
  - SET DEFAULT: Set value of foreign key to default
  - CASCADE: Propagate z to foreign key

Specifying Actions

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 )

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.

Triggers

- We saw examples of how a user can specify the reaction to a violation of a foreign key constraint
- Trigger: a more general mechanism for handling events
  - Not in SQL-92, but is in SQL-1999
- Trigger is a schema element (like table, assertion, …)
- Basic idea:
  - Whenever a specified event occurs
  - If a condition is true
  - Execute some specified action
Triggers (cont’d)

• Triggers allow definition of a wider variety of constraint violations
  – E.g., inclusion dependency
• Example: When CrsCode or Semester is changed, the Grade must be null
  – I.e., if a Grade is already assigned, CrsCode or Semester cannot be changed

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

• We will look at triggers in more detail later in the course

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 physically stored, constructed when accessed
  • Alternate terminology for “not physically stored”: not materialized
  – Similar to a subroutine in ordinary programming

Views - Examples

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

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:

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

Views: Examples (cont’d)

• A view may be defined over more than 1 tables
• The names of view attributes can be different from the attributes of the underlying tables

CREATE VIEW Professor_SStudent (Prof, Students) AS
SELECT Teaching.ProfId, Transcript.StudId
FROM Transcript, Teaching
WHERE Transcript.CrsCode = Teaching.CrsCode
AND Transcript.Semester = Teaching.Semester

• Although views are like tables, not all views are updateable
  • Examples to come later in the course
Modifying the Schema

ALTER TABLE Student
ADD COLUMN Gpa INTEGER DEFAULT 0

ALTER TABLE Student
ADD CONSTRAINT GpaRange -- constraints can be named
CHECK (Gpa >= 0 AND Gpa <= 4)

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

DROP TABLE Employee

DROP ASSERTION DontFireEveryone

Table Drop & FK Violation

• Dropping the referenced table would cause foreign key violation at the schema level
• This happens regardless of:
  – The strategy defined for handling FK violation at the instance level
  – Whether or not the table(s) have any rows

Table Drop & FK Violation (cont’d)

• Default: DROP TABLE statement is rejected
• If CASCADE CONSTRAINTS clause is specified, the referenced table is dropped and the foreign key constraint in the referring table is removed
• Rows in referring table are NOT modified

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)
Owner of Schema Object

- Each database object is owned by a particular database user
- E.g., if I log in as user Nauman and create a table Employee, then that table is owned by Nauman
- The table can be referenced by using the owner name with the table name with the syntax: Nauman.Employee
  - E.g. SELECT name from nauman.employee;
- But not everyone should be allowed all types of access to every table

Controlling Authorization in SQL

GRANT access_list ON table TO user_list
- 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

Controlling Column Authorization Using Views

- But SELECT access control to individual columns can be simulated through views (next)
- Example:
  - For table Student to allow access to only columns Name and Status create a view and then give access to that view
    CREATE VIEW Student_info AS SELECT name, status FROM Student
    GRANT SELECT ON Student_info TO joe

Authorization Mode REFERENCES

- Foreign key constraint enforces relationship between tables that can be exploited
- Example 1: prevent deletion of rows using foreign key
  CREATE TABLE DontDismissMe (
    Id INTEGER,
    FOREIGN KEY (Id) REFERENCES Student ON DELETE NO ACTION )
- Now a student referenced in DontDismissMe table cannot be deleted from Student table
Authorization Mode REFERENCES (cont’d)

- Example 2: Can be used to Reveal information
  - successful insertion into DontDismissMe means a row with foreign key value exists in Student

  `INSERT INTO DontDismissMe ('111111111')`

- Use of an attribute in a foreign key requires GRANT in REFERENCES mode

  `GRANT REFERENCES ON Student TO joe`

Summary

- Data model is a collection of conceptual tools for describing:
  - Data
  - Data relationships
  - Data semantics
  - Consistency constraints

- Having data models abstracted at different levels provides data independence
  - Physical and conceptual data independence

- Relational data model provides conceptual tools for conceptual and external levels

Summary (cont’d)

- Relational data model based on the mathematical notion of relations
  - Relation schema and instance
  - Definition of various types of constraints
    - Key, foreign key, not null

- SQL support definition of schema (via DDL)
  - `CREATE/ALTER/DROP`
    - `TABLE, CONSTRAINT, TRIGGER, VIEW, DOMAIN, ASSERTIONS`
  - Authorization support provided via GRANT (and REVOKE)