The Relational Data Model and SQL Data Definition Language

Paul Fodor

CSE316: Fundamentals of Software Development

Stony Brook University

http://www.cs.stonybrook.edu/~cse316

The *Relational* Data Model

- A particular way of structuring data (using *relations*) proposed in 1970 by E. F. Codd
  - Mathematically based
  - Expressions (≡ *queries*) can be analyzed by DBMS and transformed to equivalent expressions automatically (*query optimization*)
    - Optimizers have limits (=> the programmer still needs to know how queries are evaluated and optimized)

Relational Databases

- In Relational DBs, the data is stored in tables

Attributes (column headers)

Relation name: Student

Tuples (rows)

Id	*Name*	*Address*	*Status*
1111	John	123 Main	fresh
2222	Mary	321 Oak	
1234	Bob	444 Pine	
9999	Joan	777 Grand	senior

Table

- Set of rows (**no duplicates**)
- Each *row/tuple* describes a different entity
- Each *column/attribute* states a particular fact about each entity
  - Each column has an associated *domain*

Id	*Name*	*Address*	*Status*
1111	John	123 Main	fresh
2222	Mary	321 Oak	
1234	Bob	444 Pine	
9999	Joan	777 Grand	senior

    - Domain of *Id*: integers
    - Domain of *Name*: strings
    - Domain of *Status* = {fresh, soph, junior, senior}

Relation

- A relation is the mathematical entity corresponding to a table:
  - A relation **R** can be thought of as predicate **R**
    - A tuple (x,y,z) is in the relation **R** iff **R**(x,y,z) is true
- Why Relations/Tables?
  - Very simple model.
  - Often matches how we think about data.
  - Underlies Excel, the most important application in industry today.

Operations

- Operations on relations are precisely defined:
  - Take relation(s) as argument, produce new relation as result
    - Unary (*e.g.*, selection, projection, delete certain rows)
    - Binary (*e.g.*, natural join, Cartesian product, union)
  - Using mathematical properties, *equivalence* of expressions can be decided (important for *query optimization*)

$$op1(T1,op2(T2)) \stackrel{?}{=} op3(op4(T1),T2)$$

- these expressions could have different execution time complexities

Relation Schema

- A relation schema is the heading of that table and the applicable constraints (integrity constraints).
- A *relation schema* is composed of:
  - Relation name
  - Attribute names & domains
  - *Integrity constraints* like:
    - The values of a particular attribute in all tuples are unique (e.g., PRIMARY KEY)
    - The values of a particular attribute are greater than 0 (although, the domain can be bigger)
  - Default values for some attributes
- A *relation* consists of a relation schema and a relation instance.

Schemas

- *Relation schema* example:
  - Example: Student(<u>id</u>, name, address, status) or Student(<u>id:int</u>, name:string, address:string, status:string{fresh, soph, junior, senior})
    - <u>Underline</u> an attribute means a *key* (tuples cannot have the same value in key attributes) - this is an example of a constraint on the relation.

Relational Database

- A *database schema* = the set of relation schemas and constraints among relations (*inter-relational* constraints)
- A *relational database* (or *database instance)* = a set of (corresponding) relation instances

Database Schema Example

- *Student (Id: INT, Name: STRING, Address: STRING, Status: STRING)*
- *Department(DeptId: STRING, Name: STRING)*
- *Professor (Id: INT, Name: STRING, DeptId: STRING)*
- *Course (CrsCode:STRING, DeptId:STRING, CrsName:STRING, Descr:STRING)*
- *Teaching (ProfId:INTEGER, CrsCode:STRING, Semester:STRING)*
- *Transcript (StudId:INTEGER, CrsCode:STRING, Semester:STRING, Grade:STRING)*

Relation Instance

- A *relation instance* is a **set** of tuples:
  - Tuple ordering immaterial
  - No duplicates
- All tuples in a relation have the same structure; constructed from the same set of *attributes*:
  - Attributes are named (ordering is immaterial)
  - Value of an attribute is drawn from the attribute's *domain*.
  - *Arity* of relation = number of attributes.

Relation Instance

- There is also a special value ***NULL*** (commonly referred to as a *null value*, *value unknown* or *undefined*), which we use a placeholder and store it in place of an attribute until more information becomes available
  - null values arise because of a lack of information
  - null values arise by design sometimes
    - For instance, the attribute MaidenName is applicable to females but not to males

Employee(<u>Id:INT</u>, Name:STRING, MaidenName:STRING)

  - We do not allow null values in certain sensitive places, such as the primary key (see later).

Relation Instance of the Student relation

A relation instance can be represented as a table: the attributes are the column headers and the tuples are the rows:

Id	*Name*	*Address*	*Status*
1111111	John	123 Main	freshman
2345678	Mary	456 Cedar	sophmore
4433322	Art	77 So. 3rd	senior
7654321	Pat	88 No. 4th	sophmore

Student

Integrity Constraints

- An *integrity constraint* (IC) is a statement about all legal instances of a database:
  - Restriction on a state (or of sequence of states) of a database (it is a part of the schema)
- Enforced/checked automatically by the DBMS
  - Protects database from errors
  - Enforces enterprise rules
- *Intra-relational* - involve only one relation
  - e.g., all Student Ids are unique (*key*)
- *Inter-relational* - involve several relations
  - e.g., the value of the attribute Id of each professor shown as Teaching a course must appear as the value of the Id attribute of some row of the table Professor (*foreign-key*)

Kinds of Integrity Constraints

- Static – restricts legal states of database
  - Syntactic (structural constraints)
    - 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 constraints – limitation on sequences of database states
    - e.g., cannot raise salary by more than 5% in one transaction

BEFORE WE Continue

- Install MySQL Community edition: https://dev.mysql.com/downloads/mysql/
  - On Mac, You must open it from Finder with right click and Open (otherwise Mac will not want to open it).
  - It will ask you for a password for the DB root account
- After you install mysql server and shell. You should also install MySQL Workbench https://dev.mysql.com/downloads/workbench/ as a client.

Kinds of Integrity Constraints

- Syntactic constraint example:

```
CREATE TABLE Student(
     ID INTEGER,
     NAME CHAR(50) NOT NULL,
     ADDRESS CHAR(50),
     STATUS CHAR(10) DEFAULT "fresh",
     PRIMARY KEY (ID)
);
```

Key Constraint

- A *key constraint* (or *candidate key*) is a sequence of attributes $A_1,\ldots,A_n$ (n=1 is possible) of a relation schema, S, with the following property:
  - 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 property: no subset of $A_1,\ldots,A_n$ is a key constraint
- Key examples:
  - 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

Key Constraint

- Student(Id:INTEGER, Name:STRING, Address:STRING, Status:STRING)
  - Key: {Id}
- Professor(Id:INTEGER, Name:STRING, DeptId:STRING)
  - Key: {Id}
- Teaching(ProfId:INTEGER, CrsCode:STRING, Semester:STRING)
  - Key: {CrsCode,Semester}
- Course(CrsCode:STRING, DeptId:STRING, CrsName:STRING, Descr:STRING)
  - Keys: {CrsCode}, {DeptId,CrsName}
- Transcript(StudId:INTEGER, CrsCode:STRING, Semester:STRING, Grade:STRING)
  - Key: {StudId,CrsCode,Semester}

Key Constraint

- One of the keys of a relation is designated as the *primary key*.

- *Superkey* - set of attributes containing a key
  - {Id, Name} is a superkey of Student

Key Constraint

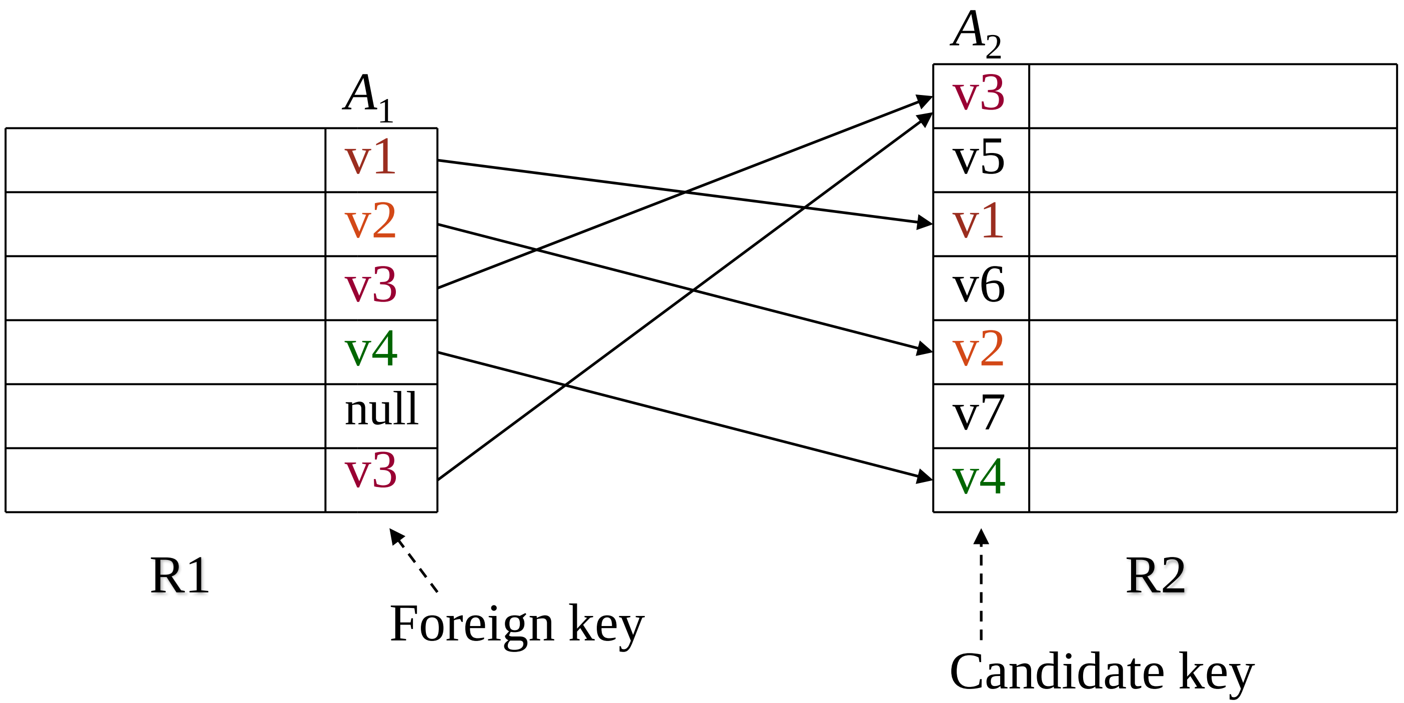
- Every relation has a key:
  - the set of all attributes in a schema, S, is always a superkey because if a legal instance of S has a pair of tuples that agree on all attributes in S, then these must be identical tuples: since relations are sets, they cannot have identical elements

Key Constraint

- A schema can have several different keys.
  - Example:
    - in the Course relation:
      - {CrsCode} can be one key.
      - {DeptId, CrsName} is also a key in the same relation because because it is unlikely that the same department will offer two different courses with the same name.

Foreign Key Constraint

- *Referential integrity*: an item named in one relation must refer to tuples that describe that item in another



Foreign Key Constraint

- Attribute A_1 is a *foreign key* of R_1 referring to attribute A_2 in R_2, if whenever there is a value v of A_1, there is a tuple of R_2 in which A_2 has value v, and A_2 is a key of R_2
  - This is a special case of referential integrity: A_2 must be a candidate key of R_2 (e.g., CrsCode is a key of Course relation)
  - If no row exists in R_2 => *violation of referential integrity*
  - Not all rows of R_2 need to be referenced: relationship is not symmetric (e.g., some course might not be taught)
  - Value of a foreign key might not be specified (DeptId column of some professor might be NULL)

Foreign Key Constraint

- Names of the attrs. in foreign key and candidate key need not be the same

Teaching(*CrsCode*: COURSES, *Sem*: SEMESTERS, *ProfId*: INT)

Professor(*Id*: INT, *Name*: STRING, *DeptId*: DEPTS)

  - ProfId attribute of Teaching references Id attribute of Professor

Foreign Key Constraint

- R_1 and R_2 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

Foreign Key Constraint

- Foreign key might consist of several columns
  - $R_1(A_1, \ldots A_n)$ references $R_2(B_1, \ldots B_n)$
    - A_i and B_i must have same domains (although not necessarily the same names)
    - $B_1,\ldots,B_n$ must be a candidate key of R_2
  - Example: (CrsCode, Semester) of Transcript references (CrsCode, Semester) of Teaching
    - Teaching(ProfId:INTEGER, CrsCode:STRING, Semester:STRING)
      - Key: {CrsCode,Semester}
    - Transcript(StudId:INTEGER, CrsCode:STRING, Semester:STRING, Grade:STRING)
      - Key: {StudId,CrsCode,Semester}

SQL

- SQL is a language for describing database schema and operations on tables
  - Data Definition Language (DDL): sublanguage of SQL for describing schema
- The SQL Keywords are case-insensitive (SELECT, FROM, WHERE, etc), but are often written in all caps.
  - The case sensitivity of the underlying operating system plays a part in the case sensitivity of database, table, and trigger names.
    - This means such names are not case sensitive in Windows, but are case sensitive in most varieties of Unix.

Data Definition Sublanguage

- An SQL schema is a description of a portion of a database

  CREATE SCHEMA SRS_StudInfo

- It usually is under the control of a single user who has the authorization to create and access the objects within it.

  CREATE SCHEMA SRS_StudInfo AUTHORIZATION Joe

- Joe can also delete the shema:

  DROP SCHEMA SRS_StudInfo

Tables

- Table: is an SQL entity that corresponds to a relation
  - SQL-92 is currently the most supported standard but is superseded by SQL:1999 (recursive queries, triggers, and some object-oriented features), SQL:2003 (SQL/XML), SQL:2006 (storing XML data in an SQL, xQuery), SQL:2008 (ORDER BY), SQL:2011(Time Periods), SQL:2016 (JSON)

Table Declaration

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

Id	*Name*	*Address*	*Status*
101222333	John	10 Cedar St	Freshman
234567890	Mary	22 Main St	Sophomore

Student

SQL Types

- Database vendors generally deviate from the standard even on basic features like the data types:
  - SQL Data Types for Various DBs:

  http://www.w3schools.com/sql/sql_datatypes.asp
- We will focus on MySQL here.

SQL CHAR Types

- CHAR(n)
  - Used to store character string value of fixed length.
  - The maximum no. of characters the data type can hold is 255 characters.
  - It"s 50% faster than VARCHAR.
  - Uses static memory allocation.
  - Good for things of the same size (e.g., signatures, keys, zips, phone, ssn)
- VARCHAR(n)
  - Used to store variable length alphanumeric data.
  - The maximum this data type can hold is up to
    - Pre-MySQL 5.0.3: 255 characters.
    - In MySQL 5.0.3+: 65,535 characters shared for the row.
  - It"s slower than CHAR.
  - Uses dynamic memory allocation.

SQL Numeric Types

- INTEGER (INT)
  - Integer numerical (no decimal). Precision 10 digits.
- SMALLINT
  - Integer numerical (no decimal). Precision 5.
- BIGINT
  - Integer numerical (no decimal). Precision 19.
- DECIMAL(p,s) (same with NUMERIC, DOUBLE)
  - Exact numerical, precision p, scale s. Example: decimal(5,2) is a number that has 3 digits before the decimal and 2 digits after the decimal
- REAL
  - Approximate numerical, mantissa precision 7
- FLOAT
  - Approximate numerical, mantissa precision 16

SQL TIME Types

- DATE
  - Stores year, month, and day values.
  - Format: YYYY-MM-DD.
  - The supported range is from "1000-01-01" to "9999-12-31".
- TIME
  - Stores hour, minute, and second values
  - Format: HH:MI:SS.
  - The supported range is from "-838:59:59" to "838:59:59".
- DATETIME
  - A date and time combination.
  - Format: YYYY-MM-DD HH:MI:SS

Other SQL Types

- BLOB
  - For BLOBs (Binary Large OBjects). Holds up to 65,535 bytes of data
- ENUM(x,y,z,etc.)
  - Let you enter a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted.
- SET
  - Similar to ENUM except that SET may contain up to 64 list items and can store more than one choice.
- XML
  - Stores XML formatted data.
  - Not in MySQL, but in DB2, Oracle, SQL Server.

Primary/Candidate Keys

```
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

```
DROP TABLE course;
```

Null

- Problem: Not all information might be known when row is inserted (e.g., a 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

```
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 in SQL

- Often 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 in SQL

- 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:
  - The semantics of the CHECK clause requires that every tuple in the corresponding relation satisfy all of the conditional expressions associated with all CHECK clauses in the corresponding CREATE TABLE statement
  - The empty relation (i.e., a relation that contains no tuples) always satisfies all CHECK constraints as there are no tuples to check.

Assertion

- 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
- Unlike the CHECK conditions that appear inside a table definition, those in the CREATE ASSERTION statement must be satisfied by the contents of the entire database rather than by individual tuples of a host table.

Assertion

- Example: the Employee table cannot be empty:

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

- If, at the time of specifying the constraint, the Employee relation is empty, the SQL standard states that if a new constraint is defined and the existing database does not satisfy it, the constraint is rejected.
  - The database designer then has to find out the cause of constraint violation and either rectify the database or amend the constraint.
- MySQL does not support ASSERTIONS.

Assertion

- Example: employees cannot earn more than their managers:

```
CREATE ASSERTION EmployeeSalariesDown
    CHECK NOT EXISTS
        (SELECT * FROM Employee, Manager
        WHERE Employee.MngrId = Manager.Id AND
              Employee.Salary > Manager.Salary ))
```

User-Defined Domains

- Possible attribute values can be specified
  - Using a CHECK constraint or
  - Creating a new domain
    - An alternative SQL way to allow the user to define appropriate ranges of values, give them domain names, and then use these names in various table definitions.
  - Domain can be used in several declarations

Domains

- Examples:

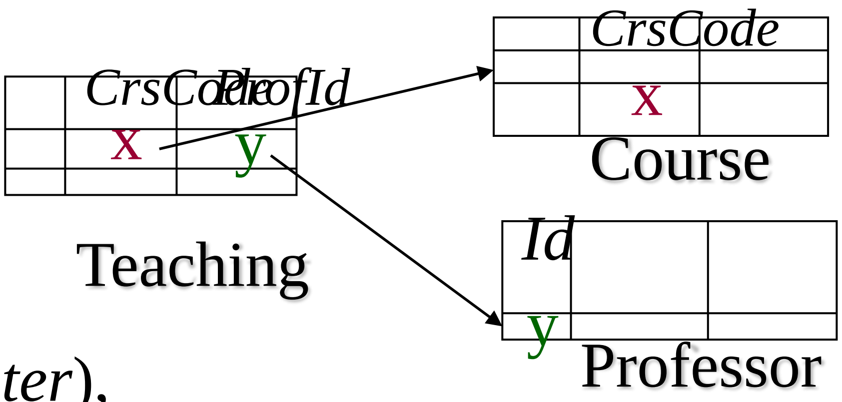
```
CREATE DOMAIN Grades CHAR (1)
   CHECK  (VALUE IN ("A", "B", "C", "D", "F"));

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

- MySQL doesn't support domains

SQL Foreign Key Constraints

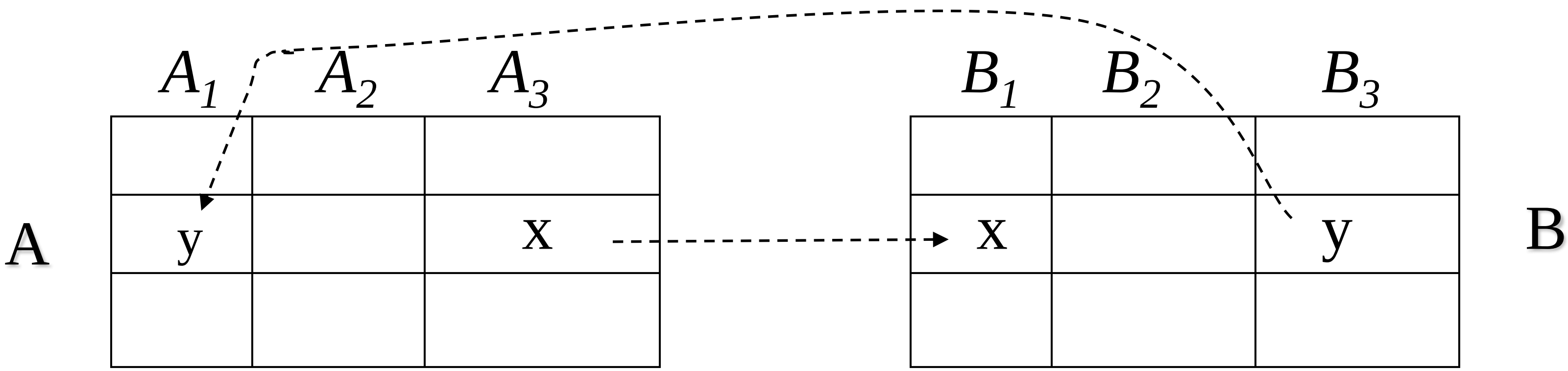
```
CREATE TABLE Teaching (
    ProfId   INTEGER,
    CrsCode   CHAR (6),
    Semester   CHAR (6),
    PRIMARY KEY (CrsCode, Semester),
    FOREIGN KEY (CrsCode) REFERENCES Course(CrsCode) ,
    FOREIGN KEY (ProfId) REFERENCES Professor (Id) );
```



- The Course table must have a CrsCode attribute and key.
- In MySQL, we must specify the attribute even if it is the same name.

FOREIGN KEY (*CrsCode*) REFERENCES Course(*CrsCode*)

Circularity in Foreign Key Constraints



candidate key: A_1
foreign key: A_3 references B(B_1)

candidate key: B_1
foreign key: B_3 references A(A_1)

- Chicken-and-egg problem:

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 Foreign Key Constraints

- 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: a check is made after each SQL statement that changes the database
- DEFERRED checking: a check is made only when a transaction commits.

```
ALTER TABLE chicken ADD CONSTRAINT chickenREFegg
    FOREIGN KEY (eID) REFERENCES egg(eID)
    INITIALLY DEFERRED DEFERRABLE;
ALTER TABLE egg ADD CONSTRAINT eggREFchicken
    FOREIGN KEY (cID) REFERENCES chicken(cID)
    INITIALLY DEFERRED DEFERRABLE;
```

Reactive Constraints

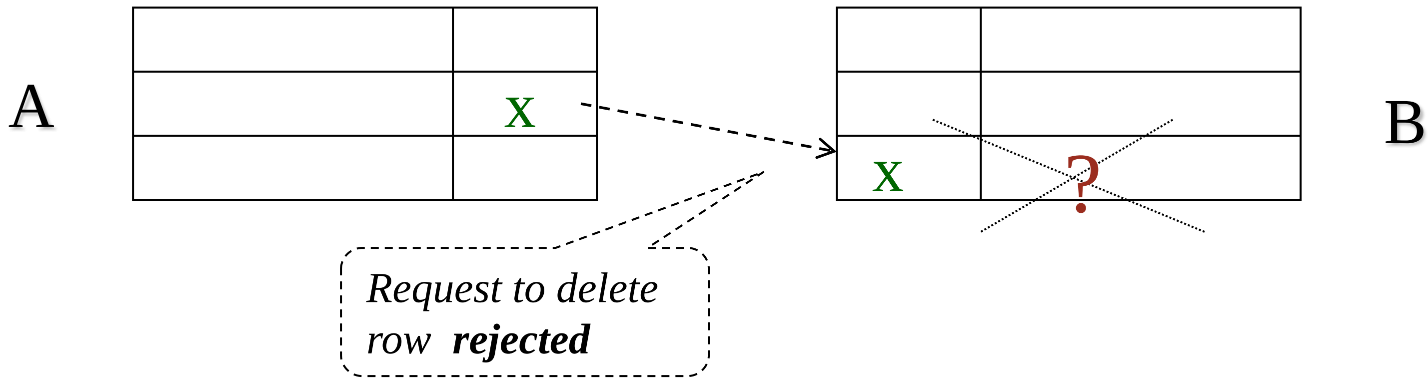
- When a constraint is violated, the corresponding transaction is typically aborted.
  - However, in some cases, other remedial actions are more appropriate.
    - Foreign-key constraints are one example of this situation.
- It would be nice to have a mechanism that allows a user to specify how to react to a violation of a constraint.
  - A reactive constraint is a static constraint coupled with a specification of what to do if a certain event happens.
    - Triggers attached to foreign-key constraints

Handling Foreign Key Violations

- Insertion into A:
  - Reject if no row exists in B containing foreign key of inserted row!

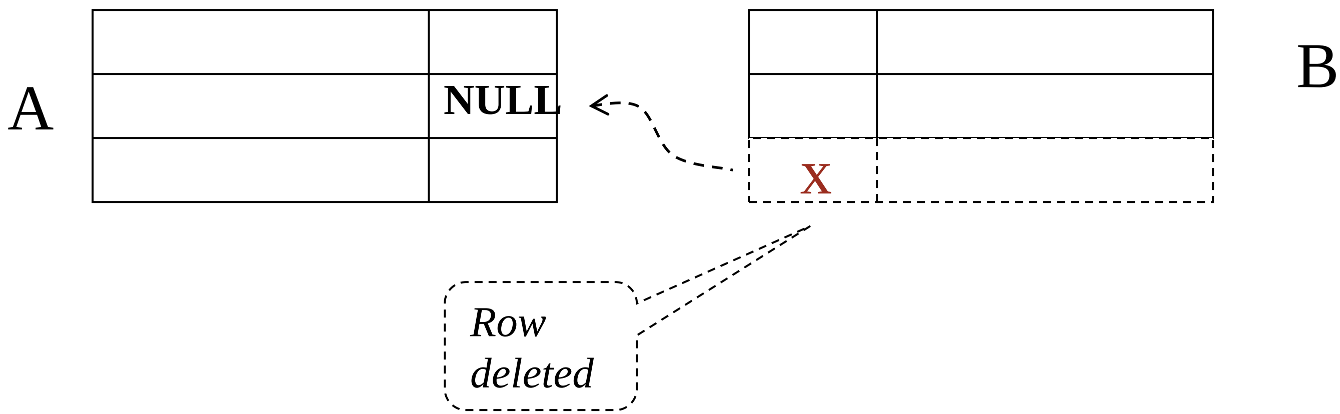
Handling Foreign Key Violations

- Deletion from B:
  - Multiple possible responses.
  - NO ACTION: Reject if row(s) in A references row to be deleted (default response)



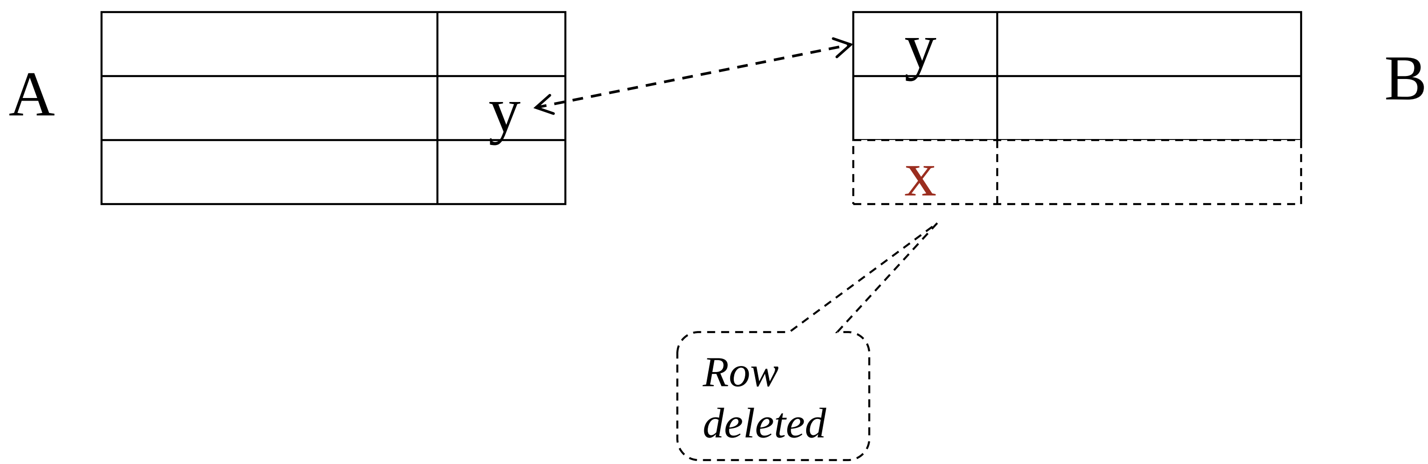
Handling Foreign Key Violations

- Deletion from B:
  - SET NULL: Set value of foreign key in referencing row(s) in A to NULL



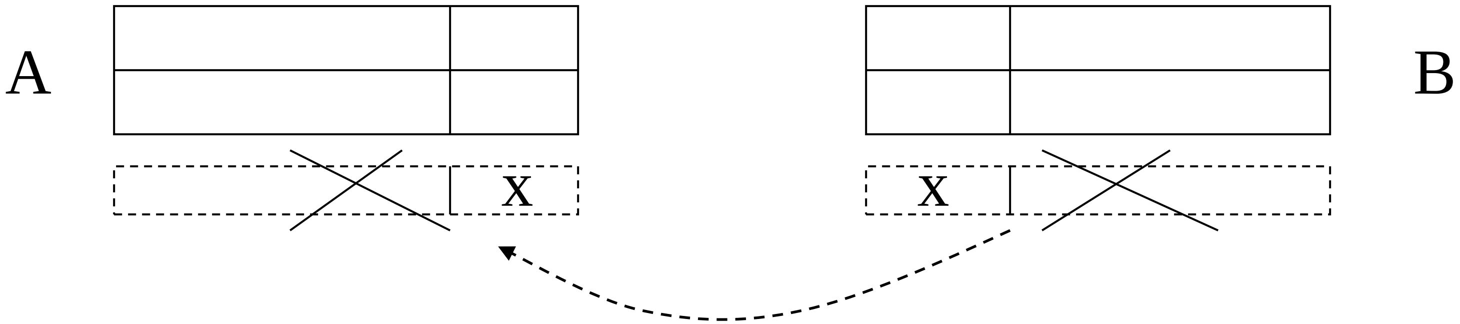
Handling Foreign Key Violations

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



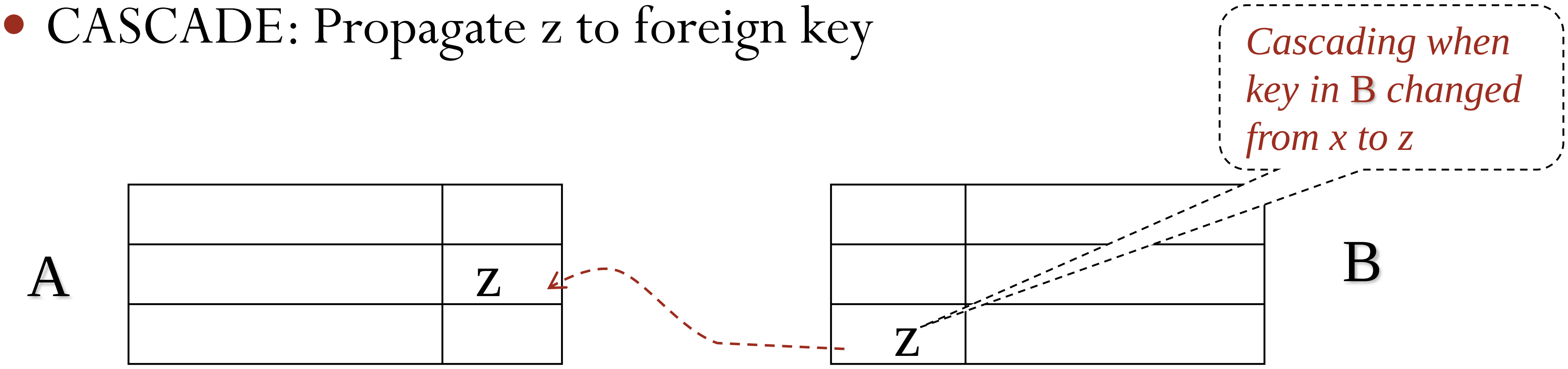
Handling Foreign Key Violations

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



Handling Foreign Key Violations

- 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



Specifying Actions

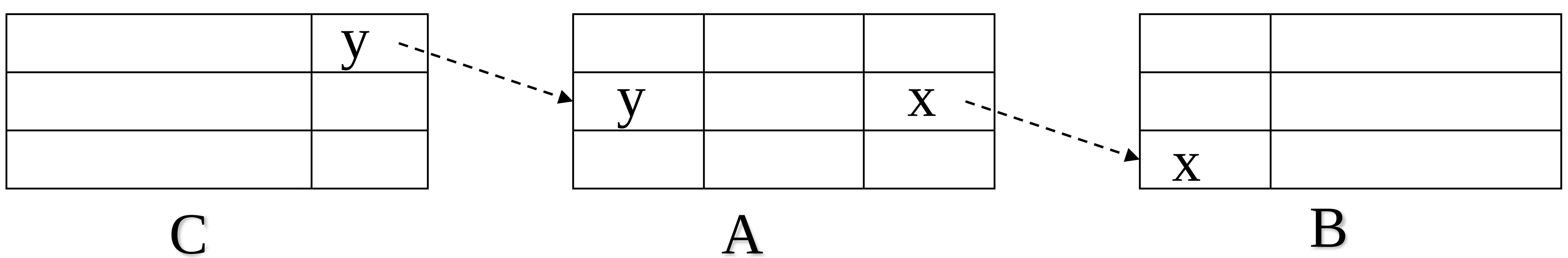
```
CREATE TABLE Teaching (
    ProfId     INTEGER,
    CrsCode CHAR (6),
    Semester CHAR (6),
    PRIMARY KEY (CrsCode, Semester),

    FOREIGN KEY (ProfId) REFERENCES Professor (Id)
        ON DELETE NO ACTION
        ON UPDATE CASCADE,

    FOREIGN KEY (CrsCode) REFERENCES Course (CrsCode)
        ON DELETE NO ACTION
        ON UPDATE CASCADE
);
```

Handling Foreign Key Violations

- 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



Insert

- Inserting a single row into a table
  - Attribute list can be omitted if it is the same as in CREATE TABLE (but do not omit it)
  - NULL and DEFAULT values can be specified

```
INSERT INTO Student(Id, Name, Address, Status)
VALUES (12345, "John Smith", "123 Main St, NYC", NULL);

INSERT INTO Student
VALUES (12345, "John Smith", "123 Main St, NYC", NULL);
```

AUTO INCREMENT

- AUTO INCREMENT – Field gets a value automatically.
  - Value is 1 more than previously added row
  - Value should never be specified in INSERT INTO

Database Views

- 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

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;
SELECT * FROM coursestaken;
```

- 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;
SELECT * FROM CoursesITook;
```

Modifying the Schema

- Although database schemas are not supposed to change frequently, they do evolve (new fields are added, etc.)

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

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

ALTER TABLE  Student
   DROP CONSTRAINT GpaRange;    -- constraint names are useful

DROP TABLE  Employee
```

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 (i.e., vendor specific)

Controlling Authorization in SQL

```
CREATE USER "joe"@"localhost" IDENTIFIED BY "password";

GRANT ALL PRIVILEGES ON * . * TO "joe"@"localhost";
```

Controlling Authorization in SQL

GRANT *access_list*
ON *table*
TO *user_list*

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

GRANT UPDATE (*Grade*) ON Transcript TO prof_smith

User name

– The *Grade* column can be updated only 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

Controlling Authorization in SQL Using Views

GRANT *access*
ON *view*
TO *user_list*

GRANT SELECT ON Course Taken TO joe

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

REVOKE

- Privileges, or the grant option for privileges, can be revoked using the REVOKE statement.

  REVOKE [GRANT OPTION FOR] privilege-list
  ON object
  FROM user-list {CASCADE | RESTRICT}

  - CASCADE means that if some user, U_1, whose user name appears on the list userlist, has granted those privileges to another user, U_2, the privileges granted to U2 are also revoked.
    - If U2 has granted those privileges to still another user, those privileges are revoked as well, and so on.
  - RESTRICT means that if any such dependent privileges exist, the REVOKE statement is rejected

REFERENCE Access mode

- The GRANT REFERENCES permission on a table is needed to create a FOREIGN KEY constraint that references that table.

GRANT REFERENCES ON Student TO joe

- Now Joe can create tables that have foreign keys to Student keys

Access mode problem

- Granting privileges at the level of database operations, such as SELECT or UPDATE, is not adequate.
  - For example, only a depositor can deposit in a bank account and only a bank official can add interest to the account, but both the deposit and interest transactions might use the same UPDATE statement.
- For such applications, it is more appropriate to grant privileges at the level of subroutines or transactions.

Triggers

- The ON DELETE/UPDATE triggers are simple and powerful, but they are not powerful enough to capture a wide variety of constraint violations that arise in database applications and are not due to foreign keys
- A more general mechanism for handling events
  - Whenever a specified event occurs, execute some specified action
  - Not in SQL-92, but is in SQL:1999

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

- Trigger is a schema element (like table, assertion, etc.)

Table Deletion

- Tables can be deleted with the DROP TABLE command
  - Use with caution! This will delete a table's definition as well as all data in rows and fields!

DROP TABLE *table_name*;

- Data in a table can be removed without deleting the table
  - Use TRUNCATE TABLE

TRUNCATE TABLE *table_name*;

Examples

```
CREATE TABLE Student (
 Id INTEGER,
 Name VARCHAR(60),
 Address VARCHAR(100),
 Status VARCHAR(20),
 PRIMARY KEY(Id)
);

CREATE TABLE Department (
 DeptId CHAR(6),
 Name VARCHAR(60),
 PRIMARY KEY(DeptId)
);

CREATE TABLE Professor (
 Id INTEGER,
 Name VARCHAR(60),
 DeptId CHAR(6),
 PRIMARY KEY(Id),
 FOREIGN KEY (DeptId) REFERENCES Department(DeptId)
);
```

Examples

```
CREATE TABLE Course (
 DeptId CHAR(6),
 CrsCode CHAR(6),
 CrsName VARCHAR(60),
 Descr VARCHAR(100),
 PRIMARY KEY(CrsCode)
)

CREATE TABLE Transcript (
 StudId INTEGER,
 CrsCode CHAR(6),
 Semester VARCHAR(20),
 Grade CHAR(2),
 PRIMARY KEY (StudId, CrsCode, Semester),
 FOREIGN KEY (StudId) REFERENCES Student(Id),
 FOREIGN KEY (CrsCode) REFERENCES Course(CrsCode)
)
```

Examples

```
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 NO ACTION
 ON UPDATE CASCADE
)
```

Examples

```
INSERT INTO Student VALUES(1,"Joe","1 Main","fresh");
INSERT INTO Professor VALUES(1, "Paul Fodor","CSE");
INSERT INTO Department VALUES("CSE", "Computer Science");
INSERT INTO Course VALUES("CSE114", "CSE", "Computer Science 1", "Procedural and Object-oriented Programming");
INSERT INTO Teaching VALUES(1, "CSE114", "Spring 2020");
INSERT INTO Transcript VALUES(1, "CSE114", "Spri20", "A");
```

Examples

```
SELECT * FROM student;
SELECT * FROM Professor;
SELECT * FROM Department;
SELECT * FROM Course;
SELECT * FROM Teaching;
SELECT * FROM transcript;
```