Integrity Checking

Translating Integrity Rules to SQL

Chuan LU
(With credit to Nigel Hardy)

Modelling Persistent Data
CS27020
Computer Science Department

Outline

Domains

- The relational model says that domains must exist
- SQL specifies a minimum list of data types
  - Varies with versions of SQL
  - `https://en.wikibooks.org/wiki/Structured_Query_Language/Data_Types`
- Each DBMS will offer these plus possibly more
  - Postgres: `https://www.postgresql.org/docs/11/datatype.html`
- Some attributes may benefit from tighter constraint on values

Attributes with constraints

- Limiting acceptable values when the table is created
  - or "ALTER"ed
- (`NOT NULL`)
- `CHECK` clauses can be added
  - to individual attributes
  - to the table, concerning attributes

Simple CHECK

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric CHECK (price > 0)
);
```

Credit: Postgresql documentation

`https://www.postgresql.org/docs/11/ddl-constraints.html`

Named constraint

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric
    CONSTRAINT positive_price CHECK (price > 0)
);
```

Credit: Postgresql documentation

`https://www.postgresql.org/docs/11/ddl-constraints.html`

Table constraints

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric CHECK (price > 0),
    discounted_price numeric CHECK (discounted_price > 0),
    CHECK (price > discounted_price)
);
```

Credit: Postgresql documentation

`https://www.postgresql.org/docs/9.6/ddl-constraints.html`

Application program constraint

- Domain constraint could be left to the application program
- Duplicate code, error prone
- Possible that constraints vary with time or circumstance
- Double check may be beneficial
  - “front end” feedback

Creating new domains

- Multiple use

```
CREATE DOMAIN name [ AS ] data_type
    [ COLLATE collation ]
    [ DEFAULT expression ]
    [ constraint [ ... ] ]

where constraint is:

[ CONSTRAINT constraint_name ]
{ NOT NULL | NULL | CHECK (expression) }
```

Credit: Postgresql documentation

`https://www.postgresql.org/docs/9.6/ddl-constraints.html`

New domain example

```
DROP DOMAIN IF EXISTS cooktemp;

CREATE DOMAIN cooktemp AS INT
  CHECK ( VALUE >= 100 AND VALUE <=250);

CREATE TABLE receipe (
  name VARCHAR(30) PRIMARY KEY,
  temp cooktemp
  -- and obviously other attributes
);

INSERT INTO receipe VALUES ('Jam Tarts',180);
INSERT INTO receipe VALUES ('Mince Pies',1800);
```

IN construct

- "enumeration"

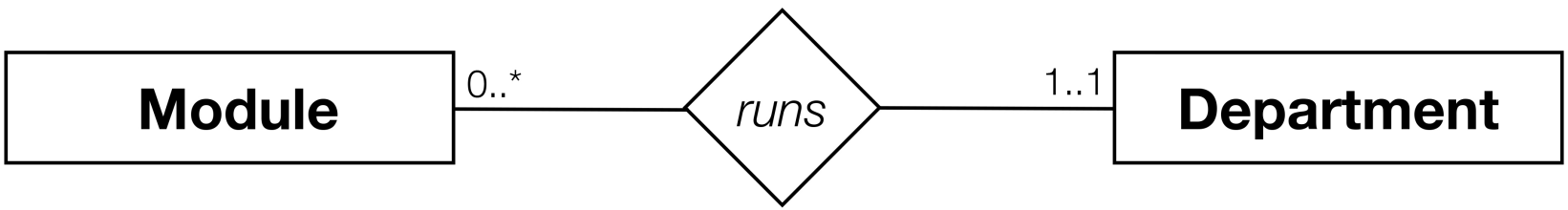
```
CREATE TABLE students (
  id VARCHAR(9) PRIMARY KEY,
  name VARCHAR(30),
  -- and other attributes
  level VARCHAR(3) CHECK IN ('UG','PGT','PGR')
);
```

- Is the list fixed permanently?
- Would an FK be better?

Outline

Outline

One to Many (Optional Many)



Integrity Rules: A module must be run by a single department; however, a department can have zero to many modules (i.e. service departments vs. academic departments).

The "many" end contains a foreign key reference to the "one" end.

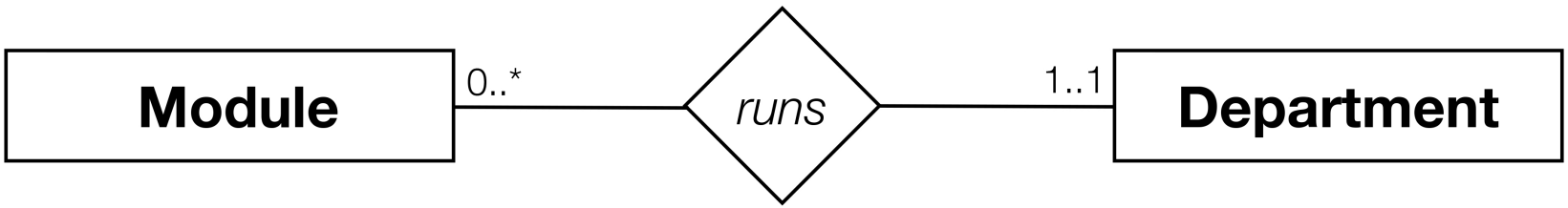
One to Many (Optional Many) - SQL implementation

```
CREATE TABLE Department(
        id VARCHAR(5) PRIMARY KEY,
        name VARCHAR(50),
        UNIQUE(id, name));


CREATE TABLE Module(
        id VARCHAR(7) PRIMARY KEY,
        deptID VARCHAR(5) NOT NULL
                REFERENCES Department(id),
        title VARCHAR(50),
        UNIQUE(id, deptID, title));
```

(70_DeptModule.sql)

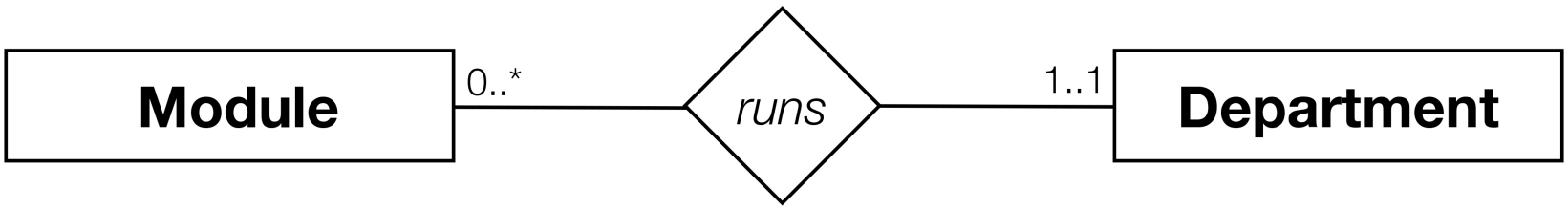
One to Many (Optional Many)



Cardinality implementations:

0	There need be no module record with a particular department PK planted
*	There can be many module records with a particular department PK planted
1	ID is a PK - it cannot be null
1	ID is a PK - it cannot be duplicated

One to Many (Optional Many)



Cardinality implementations:

0	There need be no module record with a particular department PK planted
*	There can be many module records with a particular department PK planted
1	ID is a PK - it cannot be null
1	ID is a PK - it cannot be duplicated

(In all these examples for “PK” read “PK or CK”)

Many to Many (Optional Both Sides)

Integrity Rules: A student may take no modules, or may take one or more modules. A module may have no registered students, or may have several.

The many-to-many relationship becomes a new relation with a composite key (the *takes* relationship becomes the Takes relation).

(StudentModule.sql)

Many to Many (Optional Both Sides) - SQL implementation

```
CREATE TABLE Student(
        studentID VARCHAR(5) PRIMARY KEY,
        firstname VARCHAR(50),
        surname VARCHAR(50));

CREATE TABLE Module(
        id VARCHAR(7) PRIMARY KEY,
        title VARCHAR(50),
        UNIQUE(id, title));

CREATE TABLE Takes (
        studID VARCHAR(5)
            REFERENCES Student(studentID),
        modID VARCHAR(7)
            REFERENCES Module(id),
        PRIMARY KEY (studID,modID)
       );
```

(80_StudentModule.sql)

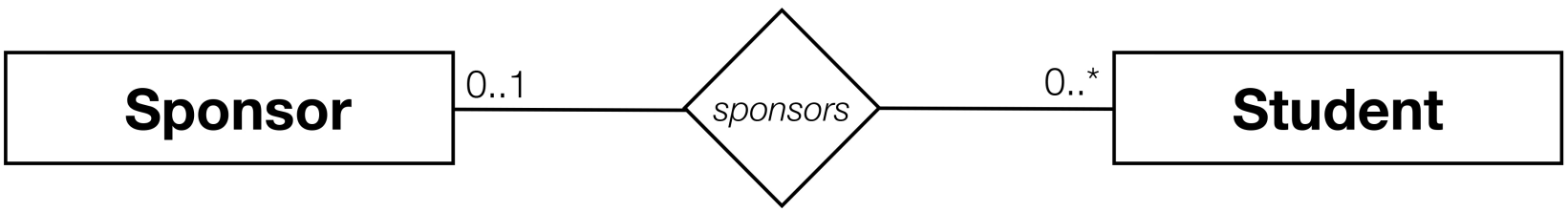
Many to Many (Optional Both Sides)

Cardinality implementations:

0	There need be no takes record with a particular module PK planted
*	There can be many takes records with a particular module PK planted
0	There need be no takes record with a particular student PK planted
*	There can be many takes records with a particular student PK planted

student cannot take the same module twice: PK of takes

One to N (Optional Both Sides)



Integrity Rules: A student may have at most one sponsor. A sponsor may have no students, or may have several.

Create a new relation with same key as 'many' end.

(StudentSponsor.sql)

One to N (Optional Both Sides) - SQL implementation

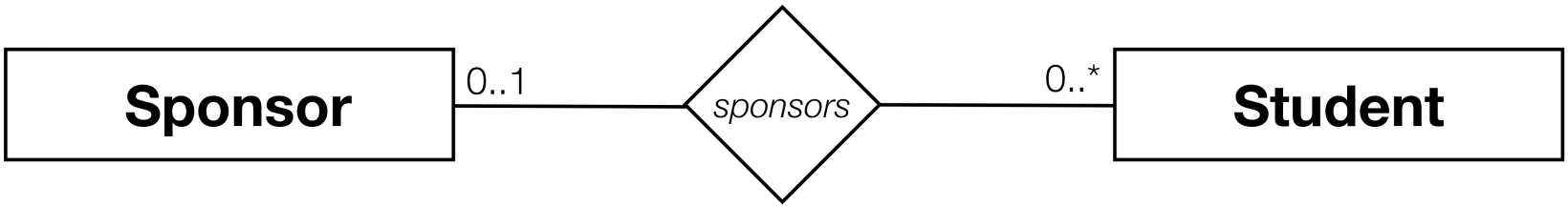
```
CREATE TABLE Student(
        studentID VARCHAR(5) PRIMARY KEY,
        firstname VARCHAR(50),
        surname VARCHAR(50));


CREATE TABLE Sponsor(
        companyID VARCHAR(6) PRIMARY KEY,
        companyName TEXT,
        UNIQUE(companyID, companyName));

CREATE TABLE Sponsors(
        studentID VARCHAR(5)
                REFERENCES Student(studentID),
        companyID VARCHAR(6)
                REFERENCES Sponsor(companyID),
        PRIMARY KEY (studentID));
```

(90_StudentSponsor.sql)

One to N (Optional Both Sides)



Cardinality implementations:

0	There need be no sponsors record with a particular student PK planted
1	There can only be 1 sponsors record with a particular student PK planted - PK (unique)
0	There need be no sponsors record with a particular sponsor PK planted
*****	There can be many sponsors records with a particular sponsor PK planted

Alternative One to N (Optional Both Sides)

Allow a `NULL` FK

Alternative One to N (Optional Both Sides)

Allow a NULL FK

```
CREATE TABLE Sponsor(
  companyID VARCHAR(6) PRIMARY KEY,
  companyName TEXT,
  UNIQUE(companyID, companyName));

CREATE TABLE Student(
  studentID VARCHAR(5) PRIMARY KEY,
  firstname VARCHAR(50),
  surname VARCHAR(50));

CREATE TABLE Sponsors(
  studentID VARCHAR(5)
    REFERENCES Student(studentID),
  companyID VARCHAR(6) NOT NULL
    REFERENCES Sponsor(companyID),
  PRIMARY KEY (studentID));
```

Alternative One to N (Optional Both Sides)

Allow a `NULL` FK

```
CREATE TABLE Sponsor(
  companyID VARCHAR(6) PRIMARY KEY,
  companyName TEXT,
  UNIQUE(companyID, companyName));

CREATE TABLE Student(
  studentID VARCHAR(5) PRIMARY KEY,
  firstname VARCHAR(50),
  surname VARCHAR(50));

CREATE TABLE Sponsors(
  studentID VARCHAR(5)
    REFERENCES Student(studentID),
  companyID VARCHAR(6) NOT NULL
    REFERENCES Sponsor(companyID),
  PRIMARY KEY (studentID));
```

```
CREATE TABLE Sponsor(
  companyID VARCHAR(6) PRIMARY KEY,
  companyName TEXT,
  UNIQUE(companyID, companyName));

CREATE TABLE Student(
  studentID VARCHAR(5) PRIMARY KEY,
  firstname VARCHAR(50),
  surname VARCHAR(50)
  sponsor VARCHAR(6)
    REFERENCES Sponsor(companyID);
```

Integrity Checking so far

- UNIQUE
- NOT NULL
- Build-in domains (types)
- CHECK
- new DOMAIN
- Referential Integrity for
  - 1:M (Optional many)
  - M:N (Optional on Both Sides)
  - 1:N (Optional on Both Sides)

Code available from Blackboard