Ian Kenny

October 27, 2016

Databases

Lecture 5

Introduction

In this lecture

- Logical design.
- Creating relational schemas.
- Constraints.
- Creating tables in SQL, with constraints.
- Inserting data into tables in SQL.

Worked example: relations

The relations resulting from our initial design are (in no particular order)

- Album.
- Artist.
- Customer.
- Sale.
- Genre.
- Review.
- Label.
- Composer
- Credit.
- Favourite.

CreditCard had a one-to-one relationship with Customer so will be made an attribute of Customer.

Logical design

In the logical design phase, we create a design that is specific to a particular database model. We will be using the relational model.

In this phase, designers continue to use ER modelling but now with the specific underlying model in mind. We will proceed on the basis of the ER model we have already created rather than creating more diagrams.

The outcome of the first part of the logical design phase for us will be the relational schemas: the schemas for each relation which tell us the attributes and the domains of those attributes, the primary keys and other constraints.

Relational schemas

For each relation identified we need to create a relational schema. A relational scheme identifies the attributes and their domains. We can also identify constraints at this stage, e.g. primary keys, etc.

Constraints review

NOT NULL: the not null constraint specifies that the attribute cannot contain null. The attribute must be given a value from the domain.

UNIQUE: the unique constraint specifies that the attribute value must be unique with respect to other values of the same attribute.

UNIQUE and NOT NULL: an attribute specified as unique and not null is a candidate key.

PRIMARY KEY: a candidate key that has been chosen as the primary key.

FOREIGN KEY

A FOREIGN KEY is an attribute in a table that references a primary key in another table. Frequently the table with the primary key is called the 'parent table' and the table with the foreign key the 'child table'. Foreign keys enable joins between tables.

The foreign key constraint simply means that if an attribute has a value that value must exist in the primary key attribute of the parent table.

FOREIGN KEY

A foreign key can be null and does not have to be unique, unless otherwise specified.

If a foreign key value can be null this means it does not have to have a value allocated to it from the primary key attribute of the parent table. If a foreign key value can be non-unique this means that multiple rows in the child table can reference the same primary key attribute in the parent table.

Constraints review

For each attribute, we can specify CHECK constraints that determine the range of values from the domain that may be used for the attribute. For example, we might need to limit an age attribute to 18 and over (CHECK (age $>=$ 18)).

Checks can be applied at the 'table level' too hence involve multiple attributes.

We can also specify DEFAULT values for attributes where appropriate, which means that any row not given a value for the attribute will be given the default value. For example, if most of the albums on sale in the music store are from the USA, we could set that as the default to save time on data entry.

Constraints review

Constraints can apply within tables and between them.

Within a table, it might be the case that a particular attribute must be unique, not null, be greater than 50, etc.

Between tables, it might be the case that an attribute in one table references another table and that constraints apply to this relationship.

Foreign key constraints create relationships between tables. Additional constraints on the relationship create the multiplicity constraints previously discussed.

Artist

The relational schema for Artist[1] might be as follows

Artist(*artistID* int, name varchar(255),
countryOfOrigin varchar(255))

Primary key: artistID
Foreign key: ∅
Not null: name
Unique: ∅
Default: ∅
Check: ∅

[1] note that these schemas may be different from the ones used previously on the module.

Album

The relational schema for Album might be as follows

Album(*albumID* int, artistID int, title varchar (255), label varchar (255), year numeric(255), genre varchar(255), price decimal(6, 2))

Primary key: albumID
Foreign key: Artist(artistid), Genre(name)
Not null: artistid, genre, title, price
Unique: $\emptyset$
Default: $\emptyset$
Check: price > 0

Album/Artist

If we consider the relationship between Album and Artist, we saw that the relationship is a 0..* many relationship from Artist$\rightarrow$Album. Each artist may have zero to many albums in the database. There is no constraint in this case. On the other hand, the relationship from Album$\rightarrow$Artist is one-to-one. Each album must be by one artist. In this case the artistid in Artist is a foreign key in Album and must not be null.

The Album table also has some 'internal constraints'. For example, that the price must not be null and must be greater than zero.

Customer

The relational schema for Customer might be as follows

Customer(*custID* int, fname varchar(255), lname varchar(255), houseNum varchar(255), postCode varchar(255), creditcard int)

Primary key: custID
Foreign key: $\emptyset$
Not null: lname, houseNum, postCode, creditcard
Unique: creditcard
Default: $\emptyset$
Check: $\emptyset$

Sale

The relational schema for Sale might be as follows

Sale(*salesRef* int, custID int, albumID int, saleDate date)

Primary key: salesRef
Foreign key: Customer(custID), Album(albumID)
Not null: custID, albumID
Unique: $\emptyset$
Default: $\emptyset$
Check: $\emptyset$

Review

The relational schema for Review might be as follows

Review(*albumID* int, *custID* int, rating int, text varchar(1024))

Primary key: {albumID, custID}
Foreign key: Customer(custID), Album(albumID)
Not null: $\emptyset$
Unique: $\emptyset$
Default: rating=1
Check: rating $>=$ 1 and rating $<=$ 5

Genre

The relational schema for Genre might be as follows

Genre(*name* varchar(255), description varchar(255))

Primary key: name
Foreign key: $\emptyset$
Not null:$\emptyset$
Unique: $\emptyset$
Default: $\emptyset$
Check: $\emptyset$

Label

The relational schema for Label might be as follows

Label(*name* varchar(255), *region* varchar(255), country varchar(255))

Primary key: {name, region}
Foreign key: ∅
Not null: country
Unique: ∅
Default: ∅
Check: ∅

Composer

The relational schema for Composer might be as follows

Composer(*composerID* int, name varchar(255))

Primary key: composerID
Foreign key: $\emptyset$
Not null: name
Unique: $\emptyset$
Default: $\emptyset$
Check: $\emptyset$

Credit

The relational schema for Credit might be as follows

Credit(*composerID* int, *albumID*, weighting int)

Primary key: $\{$composerID, albumID$\}$
Foreign key: Composer(composerID), Album(albumID)
Not null: weighting
Unique: $\emptyset$
Default: $\emptyset$
Check: weighting $>=$ 0 and weighting $<=$ 100

Favourite

The relational schema for Favourite might be as follows

Favourite(custid int, artistname varchar(255))

Primary key:$\emptyset$
Foreign key: Customer(custID)
Not null: custID, artistname
Unique: $\emptyset$
Default: $\emptyset$
Check: $\emptyset$

Creating the tables

We now need to actually create the tables in our chosen DBMS. (This is, strictly speaking, part of the physical design).

This consists of writing an SQL CREATE TABLE command for all of the tables we have identified.

The basic form of the CREATE TABLE command is

```
CREATE TABLE table_name(
column_name1 data_type(size),
column_name2 data_type(size),
column_name3 data_type(size),
....);
```

Artist

```
CREATE TABLE artist(
artistID int PRIMARY KEY,
name varchar(255) NOT NULL,
countryOfOrigin varchar(255));
```

Note the PRIMARY KEY constraint, and the NOT NULL constraint.

Album

```
CREATE TABLE album(
albumID int PRIMARY KEY,
artistID int REFERENCES artist(artistID) NOT NULL,
title varchar (255) NOT NULL,
label varchar (255),
year int,
genre varchar(255) REFERENCES genre(name),
price decimal(6, 2) NOT NULL CHECK (price > 0));
```

This table has a foreign key from the Artist table hence has a REFERENCES constraint for that. In this case, the artistID is also NOT NULL hence it must exist and it must exist in the Artist table. There is another foreign key from the Genre table. There is a CHECK constraint to ensure that the price has been set to greater than zero, or you could end up owing the customer money or giving albums away ...

Genre

```
CREATE TABLE genre (
name varchar (255) PRIMARY KEY,
description varchar (255));
```

Customer

```
CREATE TABLE customer(
custID int PRIMARY KEY,
fname varchar(255),
lName varchar(255) NOT NULL,
houseNum varchar(255) NOT NULL,
postCode varchar(255) NOT NULL,
creditcard int NOT NULL UNIQUE);
```

The creditcard attribute is declared as NOT NULL UNIQUE. This means it must exist and its value must be unique for that attribute. This makes creditcard a candidate key that could have been selected as the primary key.

Sale

```
CREATE TABLE sale(
salesRef int PRIMARY KEY,
custID int REFERENCES customer(custID) NOT NULL,
albumID int REFERENCES album (albumID) NOT NULL,
saleDate date NOT NULL);
```

Review

```
CREATE TABLE review(
albumID int REFERENCES album(albumID) NOT NULL,
custID int REFERENCES customer(custID) NOT NULL,
rating int NOT NULL,
text varchar (1024),
PRIMARY KEY (albumID, custID));
```

Note that this table has a composite primary key (more than one attribute) so it must be specified as shown - as a separate clause.

Label

```
CREATE TABLE label(
name varchar(255),
region varchar(255),
country varchar(255) NOT NULL,
PRIMARY KEY(name, region));
```

Composer

```
CREATE TABLE composer(
composerID int PRIMARY KEY,
name varchar(255) NOT NULL);
```

Credit

```
CREATE TABLE credit(
composerID int REFERENCES composer(composerID),
albumID int REFERENCES album(albumID),
weighting int CHECK (weighting >= 0 and weighting <= 100),
PRIMARY KEY(composerID, albumID));
```

Favourite

```
CREATE TABLE favourite(
custid int REFERENCES customer(custid),
artistName varchar(255));
```

As a weak entity, this table has no primary key.

Inserting data

Once we have created the tables, we can insert data into them. The following slides contain some examples of data being added to the tables created here in the music database.

If constraints are specified, the DBMS will automatically check those when data is added.

The PRIMARY KEY constraint is the same as 'UNIQUE and NOT NULL', therefore any value added in that column must be unique and cannot be null. The REFERENCES constraint specifies that the column that is doing the referencing must contain a value from the referenced column (usually a primary key). Hence data inserted in the foreign key attribute of the child table must exist in the referenced attribute of the parent table. Additionally, a foreign key may be declared as NOT NULL and UNIQUE. These will also be checked if present. They are not the default.

Attributes that do not reference another table can also be declared with constraints. These will simply be checked upon insertion, so NOT NULL, UNIQUE and check constraints.

Inserting data

The basic form of the SQL INSERT INTO command is

```
INSERT INTO table_name
VALUES (value1,value2,value3,...),
(value1,value2,value3,...),
...);
```

The VALUES clause need only be specified once.

Inserting data: Artist

```
INSERT INTO artist (
VALUES (0, 'David Bowie', 'UK'),
(1, 'The Beatles', 'UK'),
(2, 'Iggy Pop', 'USA'),
(3, 'Roxy Music', 'UK'),
(4, 'Kraftwerk', 'Germany'),
(5, 'Black Sabbath', 'UK'));
```

To confirm this data has been inserted correctly, you need to look at the schema for the Artist table.

Inserting data: Sale

```
INSERT INTO sale (
VALUES (1, 0, 1, to_date('2015-03-10', 'YYYY-MM-DD')),
(2, 2, 0, to_date('2015-10-03', 'YYYY-MM-DD')),
(5, 1, 8, to_date('2015-09-09', 'YYYY-MM-DD')),
(7, 7, 1, to_date('2015-07-20', 'YYYY-MM-DD')),
(9, 8, 6, to_date('2015-03-06', 'YYYY-MM-DD')));
```

The above example demonstrates the use of the *Postgres* to_date() function which formats a string as a date for use in the database.