
MONTANA
STATE UNIVERSITY

DDL - Data Definition Language

•••

Constraints On Data

Last Lecture

- DDL: Data Definition Language
  - A language that defines your relational schema
- We discussed how to define tables and columns with specific data types

```
CREATE TABLE albums(
    AlbumId  INTEGER,
    Title    NVARCHAR(160),
    ArtistId INTEGER
);
```

Constraints

- A column type is a *constraint* on the column
  - Much the same way that a statically typed language like Java places constraints on the types of variables
  - Recall that SQLite is pretty loosey goosey with types
- Most DBMS allow you to place additional constraints on columns

```
CREATE TABLE albums(
    AlbumId  INTEGER,
    Title    NVARCHAR(160),
    ArtistId INTEGER
);
```

Primary Key

- Recall that a *Primary Key* is a column or group of columns used to uniquely identify a row within a table
- By adding the PRIMARY KEY constraint to a column you are declaring that that column is the primary key for the table
    - It has a *unique value* for each row in the table

```
CREATE TABLE albums_bak (
    AlbumId  INTEGER PRIMARY KEY,
    Title    NVARCHAR(160),
    ArtistId INTEGER
);
```

Primary Key - Multiple Cols

- If you wish to declare multiple columns to be the primary key for a table, you can use the PRIMARY KEY(...) syntax
- Rare in practice, given the prevalence of synthetic keys

```
CREATE TABLE albums_bak(
    AlbumId  INTEGER,
    Title    NVARCHAR(160),
    ArtistId INTEGER,
    PRIMARY KEY (AlbumId, Title)
);
```

Primary Key - Nulls

- The SQL Specification states that primary keys must not be null
- For historical reasons, SQLite does not enforce this
- You must also add a NOT NULL constraint to the key column to get the specification behavior

```
CREATE TABLE albums_bak (
    AlbumId  INTEGER NOT NULL PRIMARY KEY,
    Title    NVARCHAR(160),
    ArtistId INTEGER
);
```

Primary Key - Nulls

- Note that the NOT NULL constraint is general and can be applied to any column
- Here it is applied to Title as well

```
CREATE TABLE albums_bak (
    AlbumId  INTEGER NOT NULL PRIMARY KEY,
    Title    NVARCHAR(160) NOT NULL,
    ArtistId INTEGER
);
```

Primary Key - SQLite

- SQLite automatically creates a *rowid* column for tables
  - Automatically gets a new unique ID for each row inserted
- If you have a single PRIMARY KEY column of type Integer, it will become an *alias* for the rowid column

```
CREATE TABLE albums_bak (
    AlbumId  INTEGER NOT NULL PRIMARY KEY,
    Title    NVARCHAR(160) NOT NULL,
    ArtistId INTEGER
);
```

Foreign Key

- Recall that a *Foreign Key* is a key in a table that *refers* to the *Primary Key* in another table
- In this table, we have an ArtistId column that refers to ArtistId in the artists table

```
CREATE TABLE albums_bak (
    AlbumId  INTEGER NOT NULL PRIMARY KEY,
    Title    NVARCHAR(160) NOT NULL,
    ArtistId INTEGER
) WITHOUT ROWID ;
```

Foreign Key Constraints

- Foreign Key (FK) constraints are important for maintaining *Referential Integrity*
- To declare a FK constraint, we use the FOREIGN KEY declaration in the table definition
  - NB: column definition is optional, PK is the default

```
CREATE TABLE albums_bak
(
    AlbumId  INTEGER       NOT NULL PRIMARY KEY,
    Title    NVARCHAR(160) NOT NULL,
    ArtistId INTEGER,
    FOREIGN KEY (ArtistId)
        REFERENCES artists (ArtistId)
) ;
```

Foreign Key Constraints

- Important Note: SQLite, again for backwards compatibility, does NOT enforce FKs by default.
- To do so, you must emit a PRAGMA statement
  - PRAGMA is a “meta” directive to the database

```
PRAGMA foreign_keys = ON;
```

Foreign Key Constraints

- UPDATE and DELETE behavior
  - You can tell a database how to behave when a value referred to by a FK is updated
  - From a practical standpoint it is extremely rare to update a value referred to by a FK
  - However *deleting rows* referred to by an FK is common

```
PRAGMA foreign_keys = ON;
```

Foreign Key Constraints

- CASCADE - when a row referred to by a FK is deleted, delete all rows in this table that referred to it
  - A delete in the Artist table *cascades* to the albums table
  - To the tracks table as well?
    - No, you need to declare it on all FKs
  - Can make deletes expensive
    - Another reason to prefer archiving

```
CREATE TABLE albums_bak
(
    AlbumId  INTEGER       NOT NULL PRIMARY KEY,
    Title    NVARCHAR(160) NOT NULL,
    ArtistId INTEGER,
    FOREIGN KEY (ArtistId)
        REFERENCES artists (ArtistId)
        ON DELETE CASCADE
) ;
```

Foreign Key Constraints

- NO ACTION & RESTRICT - Do not allow a delete from the referred to table if any row has an FK pointing to it
- You must programmatically remove all data in the appropriate order
  - I prefer this approach

```
CREATE TABLE albums_bak
(
    AlbumId  INTEGER       NOT NULL PRIMARY KEY,
    Title    NVARCHAR(160) NOT NULL,
    ArtistId INTEGER,
    FOREIGN KEY (ArtistId)
        REFERENCES artists (ArtistId)
        ON DELETE RESTRICT
) ;
```

Foreign Key Constraints

- SET NULL - On a delete on the referred to table, set the value of the FK to null for all rows that refer to that row
  - Might be useful for something like Genre, where if you remove a Genre then Tracks revert to null as “Unknown”

```
CREATE TABLE albums_bak
(
    AlbumId  INTEGER       NOT NULL PRIMARY KEY,
    Title    NVARCHAR(160) NOT NULL,
    ArtistId INTEGER,
    FOREIGN KEY (ArtistId)
        REFERENCES artists (ArtistId)
        ON DELETE SET NULL
) ;
```

Foreign Key Constraints

- SET DEFAULT - On a delete on the referred to table, set the value of the FK to the default value specified for that row

```
CREATE TABLE albums_bak
(
    AlbumId  INTEGER       NOT NULL PRIMARY KEY,
    Title    NVARCHAR(160) NOT NULL,
    ArtistId INTEGER DEFAULT 1,
    FOREIGN KEY (ArtistId)
        REFERENCES artists (ArtistId)
        ON DELETE SET DEFAULT
) ;
```

Unique Constraints

- You might use UNIQUE constraints to enforce sensible data in non-PK columns
  - Good example: email
- Shows a weakness of the synthetic key model:
  - It’s possible to get nonsense data if you don’t use additional constraints

```
CREATE TABLE albums_bak
(
    AlbumId  INTEGER NOT NULL PRIMARY KEY,
    Title    NVARCHAR(160) UNIQUE,
    ArtistId INTEGER
) ;
```

General Constraints

- SQLite allows you to define general logical checks on your data using the CHECK() clause
- The expression cannot contain a sub-query
  - This makes it of limited use in most practical cases

```
CREATE TABLE albums_bak
(
    AlbumId  INTEGER NOT NULL PRIMARY KEY,
    Title    NVARCHAR(160) UNIQUE,
    ArtistId INTEGER CHECK ( ArtistId > 0 )
) ;
```

Auto Increment

- You can declare a column to AUTOINCREMENT
- The algorithm is slightly different than *rowid*
  - It does not reuse IDs
  - It is a bit more CPU intensive
- SQLite recommends against using it

```
CREATE TABLE albums_bak
(
    AlbumId  INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT ,
    Title    NVARCHAR(160) UNIQUE,
    ArtistId INTEGER CHECK ( ArtistId > 0 )
) ;
```

DDL - Constraints

- Today we took a look at various constraints you can place on columns in your DDL
- The most important, by far, is the ForeignKey constraint (FK)
    - Delete behavior matters!
- There are also ways to constrain nulls, uniqueness, etc.
- In most systems, data integrity is maintained at both the DB and the application level
    - You are not going to be verifying email formats, for example, in the DB


MONTANA
STATE UNIVERSITY