



**MONTANA**  
**STATE UNIVERSITY**

# DDL - Data Definition Language

...

Defining Tables & Views

# Defining Tables

- DDL: Data Definition Language
  - A language that defines your relational schema
- Includes syntax for creating, modifying and deleting tables, views, indexes, etc.

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

---

# Defining Tables

- CREATE TABLE - creating tables in a database
- Syntax:

```
CREATE TABLE <name> (  
    <col name> <col type>...  
    ...  
)
```

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

---

# Defining Columns

- Column Definitions
  - A unique name within the table
  - A type
  - Additional metadata about the column

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

---

# Column Types

- Databases typically support many types of data
  - Integers
  - Strings
  - Decimal Numbers
  - Dates
  - Booleans
  - Binary (Blobs)

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

---

# Column Types - Integer

- Integer types will often specify a size (number of bytes)
  - INT
  - SMALLINT
  - BIGINT
- In SQLite these all map to the type INTEGER
  - 64 bit (8 byte) signed int

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

---

# Column Types - String

- Like integers, string types will often specify a size
  - CHARACTER(20)
  - VARCHAR(255)
  - NVARCHAR(100)
- In SQLite these all map to the type TEXT
  - No size limit even if you specify one with VARCHAR

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

---



# Column Types - Decimal

- Decimal types usually specify a precision and scale
  - DECIMAL(5, 2)
  - NUMERIC(10, 5)
- Precision: total number of decimal numbers to store
- Scale: number of decimal numbers to the right of the decimal point

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

---

# Column Types - Decimal

- Note that database can have very high precision decimal numbers
  - May be “lossy” when converting to things like a double

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

---

# Column Types - Decimal

- SQLite has two decimal related types:
  - REAL - double precision floating point number
  - NUMERIC - I *believe* double precision as well, hard to tell

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

---

# Column Types - Dates

- Typically multiple Date types
  - DATE
  - DATETIME
  - TIMESTAMP
- Careful with timezones!
- Typically stored in GMT
- SQLite stores these as NUMERIC

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

---

# Column Types - Booleans

- Usually
  - BOOLEAN
- SQLite, again, stores these as NUMERIC
  - KISS

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

---

# Column Types - Blobs

- Used to store raw binary data
  - BLOB
- SQLite actually has a BLOB data type
- A common alternative:
  - Write to disk or the cloud
  - Store URL for blob
  - Databases usually aren't very good with binary data

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

---

# Altering Tables

- ALTER TABLE statement
- Syntax:

ALTER TABLE <name>  
    <alterations>

- Here is a table rename

```
ALTER TABLE  
albums_bak  
RENAME TO albums_backup;
```

---

# Altering Tables - Add Col

- To add a column you use the ADD COLUMN clause
- Common operation as your application grows
- Some web frameworks manage these sorts of changes with a *migration management system*

```
ALTER TABLE  
albums_backup  
ADD COLUMN NEW_COL TEXT;
```

---



# Altering Tables - Rename Col

- To rename a column you use the RENAME COLUMN clause
- Uncommon in my experience

```
ALTER TABLE  
albums_backup  
RENAME COLUMN NEW_COL to NewColumn;
```

---

# Altering Tables - Drop Col

- Some databases support a DROP COLUMN clause
- SQLite does not
- You will need to
  - create a copy table
  - move data to the copy
  - rename tables
  - The ol' LeadDyno gambit

```
ALTER TABLE  
albums_backup  
DROP COLUMN NewColumn;
```

---

# Dropping Tables

- SQLite does support dropping tables
- Careful!
- Remember to sanitize all user input!

```
DROP TABLE albums_backup;
```

---

# DDL - Tables & Views

HI, THIS IS  
YOUR SON'S SCHOOL.  
WE'RE HAVING SOME  
COMPUTER TROUBLE.



OH, DEAR - DID HE  
BREAK SOMETHING?  
IN A WAY - )



DID YOU REALLY  
NAME YOUR SON  
Robert'); DROP  
TABLE Students;-- ?



OH, YES. LITTLE  
BOBBY TABLES,  
WE CALL HIM.

WELL, WE'VE LOST THIS  
YEAR'S STUDENT RECORDS.  
I HOPE YOU'RE HAPPY.



AND I HOPE  
YOU'VE LEARNED  
TO SANITIZE YOUR  
DATABASE INPUTS.

# Views

- A *View* is a result set of a stored query
- A view allows you to embed a query directly in the database
  - The DBMS may be able to optimize access to this data
  - Can encapsulate complex queries and joins to simplify data access
- Views are *Read Only*

```
CREATE VIEW tracksPlus AS
SELECT tracks.*,
       albums.Title as AlbumTitle,
       artists.Name as ArtistName
FROM tracks
      JOIN albums ON
         tracks.AlbumId = albums.AlbumId
      JOIN artists ON
         albums.ArtistId = artists.ArtistId;
```

---

# Views - Creating

- We use the CREATE VIEW statement to create views
- Here we are doing some joins to display more friendly data when we look at tracks

```
CREATE VIEW tracksPlus AS
SELECT tracks.*,
       albums.Title as AlbumTitle,
       artists.Name as ArtistName
FROM tracks
      JOIN albums ON
         tracks.AlbumId = albums.AlbumId
      JOIN artists ON
         albums.ArtistId = artists.ArtistId;
```

---

# Views - Using

- You can run queries against views
- Here we find all tracks by AC/DC in this view

```
SELECT *  
from tracksPlus  
WHERE ArtistName = "AC/DC";
```

---

# Views - Deleting

- Same as tables

```
DROP VIEW tracksPlus;
```

---



# Views

- In my experience, database administrators love views
  - Let's them create the SQL *just right...*
- Developers not so much
  - Inflexible
  - Difficult to update

```
CREATE VIEW tracksPlus AS
SELECT tracks.*,
       albums.Title as AlbumTitle,
       artists.Name as ArtistName
FROM tracks
      JOIN albums ON
         tracks.AlbumId = albums.AlbumId
      JOIN artists ON
         albums.ArtistId = artists.ArtistId;
```

---

# DDL - Tables & Views

- We looked at how to create tables with the CREATE TABLE statement
- We looked at the common database column types
  - SQLite has a simplified set of data types
  - Commercial database have much more extensive data types available
- We looked at how to alter and delete tables and columns
- Finally, we took a look at Views
  - A way to encapsulate queries in a database so that they look like tables



**MONTANA**  
**STATE UNIVERSITY**