Database Management for Beginners

Understanding the methodology and code behind database structure and management

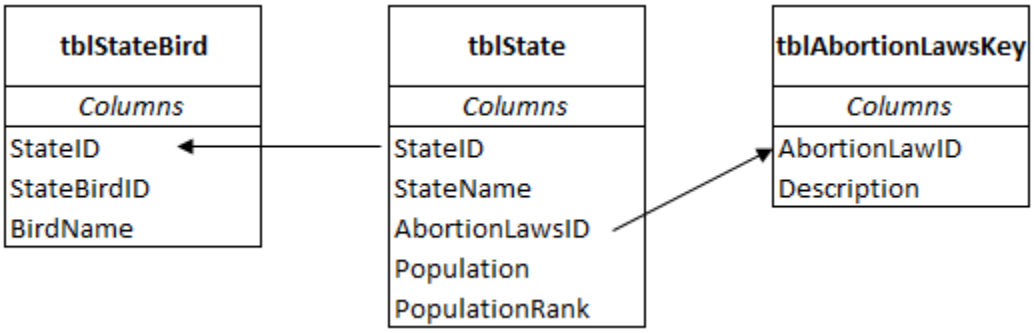
Disclaimer

The purpose of this tutorial is simply to provide an introduction to a variety of topics in Microsoft SQL. It is in no way a substitute for a full for-credit course on SQL, and should students want to pursue SQL on a more serious level they should look to the university courses on the subject

Why am I learning this? One possible career path for those with a passion for analytics and data is **Database Administration**. A database administrator (DBA) is responsible for the performance, integrity and security of a database. They will also be involved in the planning and development of the database, as well as troubleshooting any issues on behalf of the users. DBA's must have computer skills, problem solving abilities, and good communications skills.

A note on the background of Databases:

Databases are defined as any structured set of data held in a computer. For very small and simple quantities of data usually a spreadsheet software like Excel can hold your data just fine. The real power of databases comes from the concept of the **relational database.** A relational database is a database structured to recognize "relations" among stored items of information. What these relations mean is that there are hard coded relationships between different data points in your model. In the example below, we have a sample database with three tables, you'll notice the arrows drawing links between columns that appear in more than table. These columns are related in the database, and therefore when we access the data we can use those relationships to extract data from more than one table at once.



While Excel can store your data, there is no way to inherently link data sets in Excel like this, you need to use a relational database software like Access or SQL. Among the corporate world SQL Server Management Studio is by far the most popular type of database manager. For the purpose of this tutorial, we will be using SQLite, a free, lightweight, alternative to a full SQL Server Management Studio. While SQLite is limited in the amount of data it can store(10 GB), it is otherwise a fully functional SQL environment.

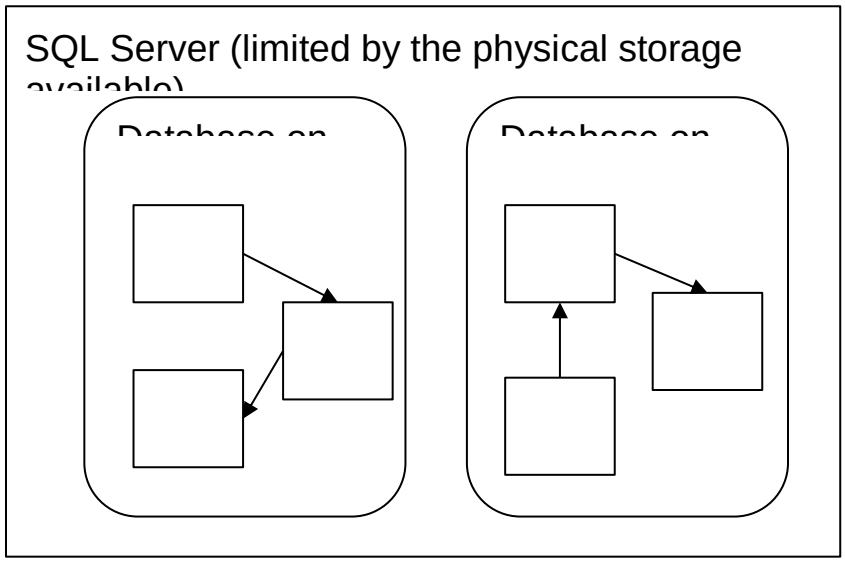
Understanding the Vocabulary

Tables: When using SQL, there is a hierarchy to data storage. The most important part of any SQL environment is the **tables**. Tables store data in columns and rows, just like excel. Every table in SQL must have a name, which usually describes the content of the table, for example a table named "customers" ideally will hold data about the customers of our company.

Databases: Tables are stored in **Databases**. Database's are simply a collection of tables that are related to each other. Like with tables, databases are named after their content. A company may have a whole database for sales, which could include tables on customers, orders, shipping, products, etc.

Servers: Finally, Databases are stored in **Servers.** Servers is where the name "SQL Server Management Studio" comes from, and they represent the physical storage of the data. When you connect to a server, you are connecting to a real world machine that is actually storing all the data for all your databases and tables. In SQLite our server is our own machine because of the data size limit, but at companies who pay for full SQL software, depending on the size of the company they could have anywhere from 1 to 10 or 20 servers. While Servers try to stick to one subject material, usually companies will try to get maximum data storage ability out of each one, which means sometimes there will be multiple types of data store in one server.

Rows and Columns: Unlike with Excel tables, SQL has a slightly stricter divide between columns and rows. While columns can be appended on to tables, it is much more difficult in SQL. As a result, it is very important when we create tables in SQL, that we carefully define every column that is in the table. Rows on the other hand, can be added to and deleted from a table with fewer consequences.



Creating a Database

For this exercise we will simply be creating the database that was used above as an example, where we include a table for states, state birds, and abortion law status.

Step 1: You must be connected to a database to do anything in SQL. For SQLite creating a database is as simple as clicking "add database" (or using the SQLite shortcut ctrl+O). In a traditional workplace environment you usually will usually not add databases unless there is a serious need, as they can take up a lot of valuable memory and storage space, but for our purposes, we can add as many as we like.

Name your database something descriptive, like "States_US"

Creating Tables

Step 2: We always start our table creation with the DROP TABLE function. This will delete any tables from our database with the name that we are trying to use. This way we avoid errors later on with possible overwriting of a table.

Code: DROP TABLE state;

*Note: the semicolon ";" is used as a separator in SQL, and allows you to run multiple pieces of code in one scripting window

Step 3: Next, we will use the CREATE TABLE function to tell SQL what we want our new table to be named and what type of data we want to include in it. The most general format for this statement is as follows:

CREATE TABLE Table_Name(
Column_Name_1 DATA TYPE,
Column_Name_2 DATA TYPE,
Column_Name_3 DATA TYPE,
PRIMARY KEY(Column_Name of primary key column)

You'll notice we've introduced some new terms here, DATA TYPE and PRIMARY KEY.

Data Types

- VARCHAR(length): Stores string data like words, names, phrases, or descriptions
- BOOLEAN: true or false data, usually represented with 0 = FALSE and 1 = True
- INTEGER or INT: Integers, frequently used for ID numbers or for units of product
- DECIMAL: Decimal numbers
- NUMERIC: Functionally the same thing as DECIMAL but formatted slightly differently
- DATE: Dates of the format YYYY-MM-DD
- DATE TIME: Combination of date and time i n the format YYYY-MM-DD hh:mm:ss.

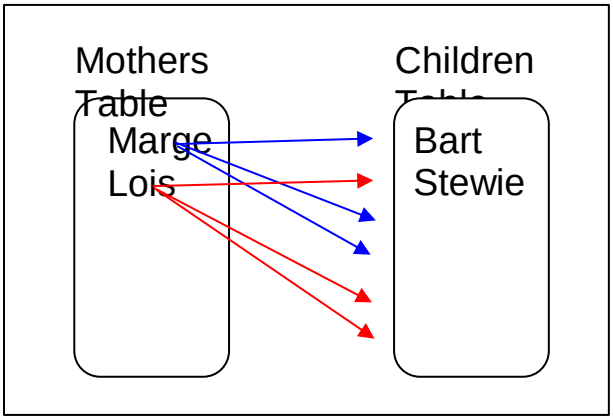
Primary Keys and the One to Many Relationship

In any given database there will be times when we care about the uniqueness of our values and times where we do not. Most of the time though, to avoid any sort of misrepresented data, it is best to set a **primary key** for your table. The primary key is a value that ensures that data that is added to a table *must be unique*. If I enter a row with primary key 167, and then I try to add

another row with the primary key 167 later on (perhaps I made a human error and forgot I had already added the data), SQL will not let you re-add the data.

The primary key also becomes very useful when understanding the relationship between tables in our relational database. A **one-to-many relationship** exists when one row in table A (for example) may be linked with many rows in table B, but one row in table B is linked to only one row in table A.

Think of it like a mother to her children:



This will be more important when it comes to querying databases, but it is an important concept to understand while we construct databases for future use.

Making our Tables

Step 4: Now that we understand this structure, we can write our code:

```
DROP TABLE State;
DROP TABLE State_Bird;
DROP TABLE Abortion_Laws_Key;

CREATE TABLE State(
State_ID INT NOT NULL,
State_Name VARCHAR,
Abortion_Laws_ID INT,
Population NUMERIC,
Population_Rank INT,
PRIMARY KEY(StateID)
);

CREATE TABLE State_Bird(
State_ID INT NOT NULL,
State_Bird_ID INT NOT NULL,
Bird_Name VARCHAR,
```

```
PRIMARY KEY(StateID)
);

CREATE TABLE Abortion_Laws_Key(
Abortion_Law_ID INT NOT NULL,
Description VARCHAR,
PRIMARY KEY(Abortion_Law_ID)
);
```

Inserting Data into our tables:
Step 5: We now have the skeleton of our tables, but no data. To add rows of data to a database there are a couple things you can do. We will be going through the method of using the INSERT INTO function, which is the most basic way of adding data. You can also add data from csv files and other database tools by using the non-coding options, but those lessons are for another time.

Code:

```
INSERT INTO State
VALUES (1, "Massachusetts", 1, 6692824, 14), (2, "California", 0, 38332521, 1), (3, "New York", 0, 19651127, 3 ), (4, "North Carolina", 1, 9848060, 10 ), (5, "Virginia", 2, 8260405, 12), (6, "Ohio", 2, 11570808, 7 ), (7, "Texas", 1, 26448193, 2 );

INSERT INTO State_Bird
VALUES (1, 4, "Chickadee"), (2, 2,"California Valley Quail"), (4, 3, "Cardinal" ), (5, 3, "Cardinal" ), (6, 3, "Cardinal" ), (3, 1, "Bluebird"), (7, 5, "Mockingbird");

INSERT INTO Abortion_Laws_Key
VALUES (0, "4 or fewer restrictions on abortion"), (1, "5 to 8 restrictions on abortion"), (2, "9 or more restrictions on abortion" );
```

Other functions for data management (Time allowing)

- UPDATE (SET)
- ALTER TABLE
- DELETE FROM

Sources for data: guttmacher.org, 50states.com, enchantedlearning.com

-Metadata: data that describes the properties and context of user data

TABLE 1-1 Example Metadata for Class Roster

Data Item	Metadata					
Name	**Type**	**Length**	**Min**	**Max**	**Description**	**Source**
Course	Alphanumeric	30			Course ID and name	Academic Unit
Section	Integer	1	1	9	Section number	Registrar
Semester	Alphanumeric	10			Semester and year	Registrar
Name	Alphanumeric	30			Student name	Student IS
ID	Integer	9			Student ID (SSN)	Student IS
Major	Alphanumeric	4			Student major	Student IS
GPA	Decimal	3	0.0	4.0	Student grade point average	Academic Unit

Entities:

Entity instance–person, place, object, event, concept (often corresponds to a row in a table)

Entity Type–collection of entities (often corresponds to a table)

Relationships:

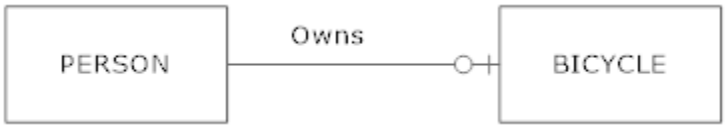
Relationship instance–link between entities (corresponds to primary key-foreign key equivalencies in related tables)

Relationship type–category of relationship…link between entity types

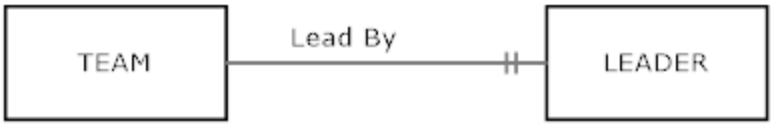
Attributes:

Properties or characteristics of an entity or relationship type (often corresponds to a field in a table)

Optional One:

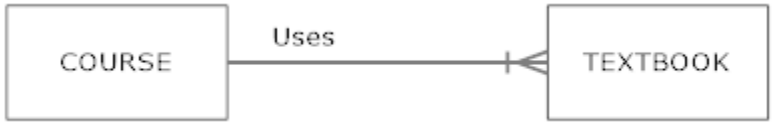


Mandatory One:



Optional Many:

Mandatory Many:



Entities:

Entity type: EMPLOYEE

Attributes	Attribute Data Type	Example Instance	Example Instance
Employee Number	CHAR (10)	642-17-8360	534-10-1971
Name	CHAR (25)	Michelle Brady	David Johnson
Address	CHAR (30)	100 Pacific Avenue	450 Redwood Drive
City	CHAR (20)	San Francisco	Redwood City
State	CHAR (2)	CA	CA
Zip Code	CHAR (9)	98173	97142
Date Hired	DATE	03-21-1992	08-16-1994
Birth Date	DATE	06-19-1968	09-04-1975

FIGURE 2-3 Entity type EMPLOYEE with two instances

Classifications of attributes:

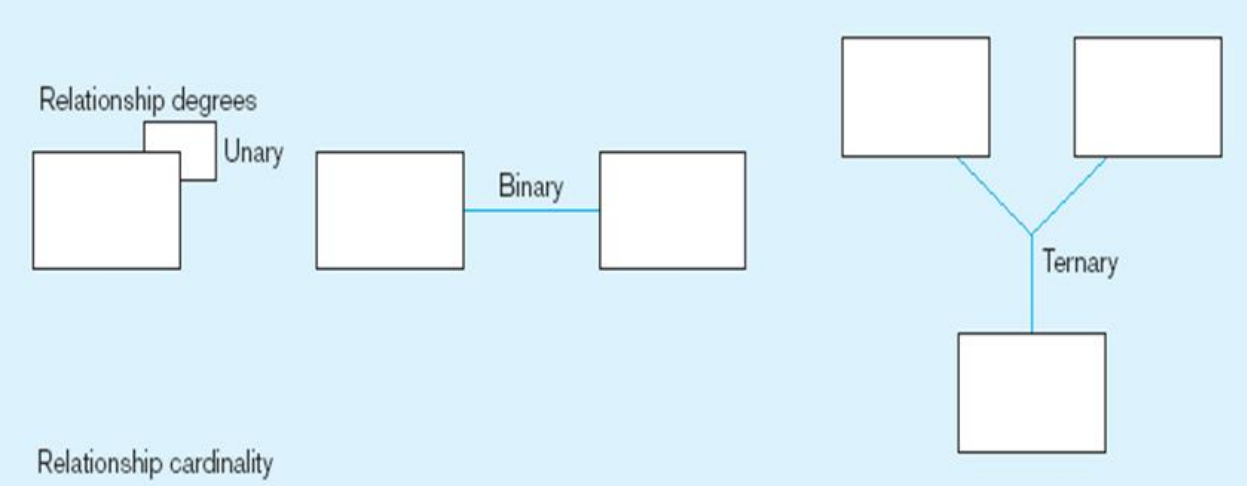
Required versus Optional Attributes

Simple versus Composite Attribute

Single-Valued versus Multivalued Attribute

Stored versus Derived Attributes

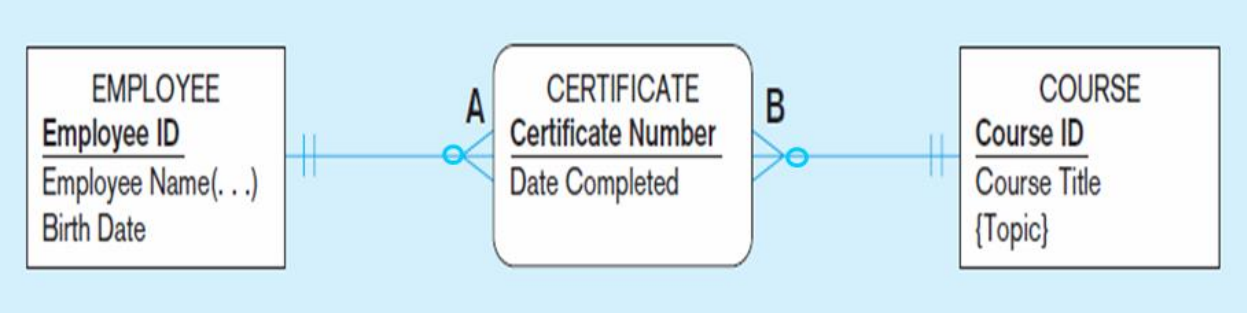
Identifier Attributes



Unary => Person married to a person

Ternary => Part, vendor and warehouse

Associative Entity



Primary keys are unique identifiers of the relation. Examples include employee numbers, social security numbers, etc. *This guarantees that all rows are unique.*

Foreign keys are identifiers that enable a dependent relation (on the many side of a relationship) to refer to its parent relation (on the one side of the relationship).

-for 1 to one relationships, key on mandatory side becomes foreign key on optional side; mandatory it does not matter

-for m:1 place foreign key on the many side of the relationship

M:N

Three Normal Forms:

1st normal form: No multivalued attributes, every attribute value must be atomic

2nd normal form: Meets 1NF and is fully functionally dependent on the entire primary key

3NF: Meets 2NF and has no transitive dependencies (dependencies on nonkey attributes)

Special Cases:

Multivalued attribute such as skill, separate table/separate relation

```
create table LISTING
        (Account_ID varchar(12),
   Rental_ID varchar(12),
        Purchase_ID varchar(12),
   Listing_ID varchar(12) primary key,
   Listing_Date varchar(10) NOT NULL,
   Price_Match_Link varchar(500) NOT NULL,
   Price decimal(5,2) NOT NULL,
   Foreign Key (Account_ID) references ACCOUNT(Account_ID),
   Foreign Key (Rental_ID) references RENTAL (Rental_ID),
   Foreign Key (Purchase_ID) references PURCHASE (Purchase_ID));
```