Module 2-5

Database Design

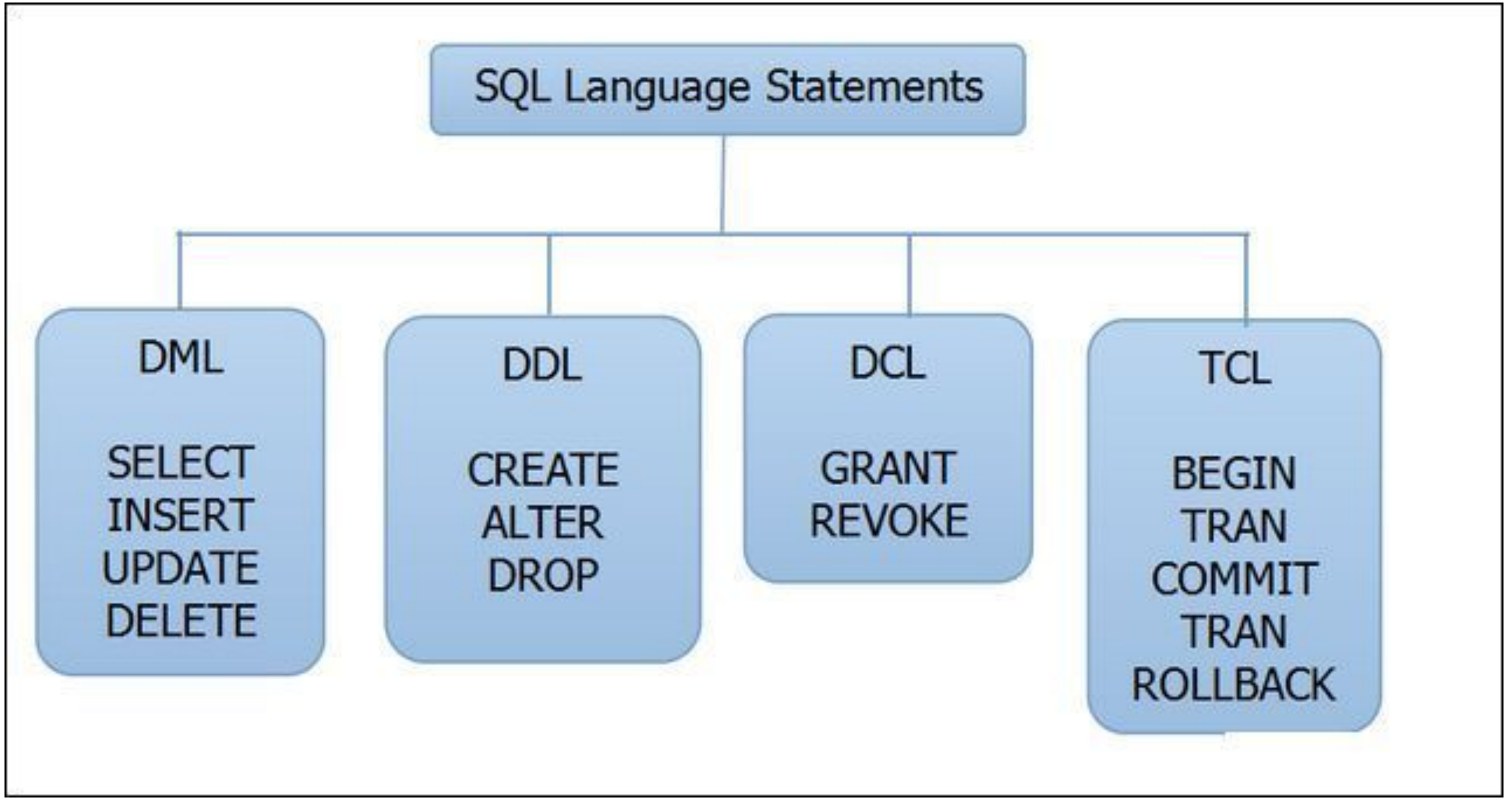
DQL vs DML vs DDL

The SQL statements we have seen so far fall into a number of different categories:

- Data Query Language (**DQL****): SELECT
- Data Manipulation Language (**DML**): INSERT, UPDATE, DELETE
- Data Definition Language (**DDL**): CREATE, ALTER

The focus of this lecture will be DDL statements with appropriate constraints.

The SQL Language Set



Database Design - Normalization

Database normalization is a process used to organize a database into tables and columns. The main idea with this is that a table should be about a specific topic and only supporting topics included.

There are three main reasons to normalize a database. The first is to minimize duplicate data, the second is to minimize or avoid data modification issues, and the third is to simplify queries.

(https://www.essentialsql.com/get-ready-to-learn-sql-database-normalization-explained-in-simple-english/)

Normal Forms

Before a single CREATE statement is run, the tables and their relationships need to be well thought out. One design methodology commonly used is 3NF, or often Boyce-Codd Normal Form (BCNF). We commonly think and talk about the first three:

First Normal Form – The information is stored in a relational table with each column containing atomic values. There are no repeating groups of columns.

Second Normal Form – The table is in first normal form and all the columns depend on the table’s primary key.

Third Normal Form – the table is in second normal form and all of its columns are not transitively dependent on the primary key

(https://www.essentialsql.com/get-ready-to-learn-sql-database-normalization-explained-in-simple-english/)

Normal Forms: 3NF

While there are several levels of “normal form” compliance, the third normal form is generally good enough for 99% of all situations.

An informal intuitive definition of 3NF is as follows:

There are no attributes of an entity (also known as a tuple) that are not directly determined by the values of the primary key when considered over the entire population of the table. In other words, one must consider the population of likely or possible values when considering the normalization.

Normal Forms: 3NF Example

Suppose we have the following table:

InvoiceNumber (PK)	InvoiceDate	Inventory ID	Inventory Description
1000	10/1/2019	45	Hammer
1001	10/3/2019	28	Nails
1002	10/3/2019	17	Screwdriver
1003	10/4/2019	45	Hammer

Some questions to consider:

- Is an invoice date directly related to an invoiceNumber? → Yes
- Is an inventory description directly related to an invoiceNumber? → No

Normal Forms: 3NF Example

Suppose we need a Spanish version of this database, and we need to value to show *Martillo* instead of Hammer. This would entail an UPDATE statement that targets 2 rows.

InvoiceNumber (PK)	**InvoiceDate**	**Inventory ID**	**Inventory Description**
1000	**10/1/2019**	**45**	Martillo
1001	10/3/2019	28	Nails
1002	10/3/2019	17	Screwdriver
1003	**10/4/2019**	**45**	Martillo

Normal Forms: 3NF Example

In this situation, we could have split up the data into 2 tables, thus we end up with a less risky query, affecting only 1 row:

InvoiceNumber (PK)	InvoiceDate	Inventory ID
1000	**10/1/2019**	**45**
1001	10/3/2019	28
1002	10/3/2019	17
1003	**10/4/2019**	**45**

Inventory ID (pk)	Description
28	Nails
17	Screwdriver
45	Martillo

Many to Many relationships

Generally speaking, when there are 2 entities for which there is a “many to many” relationship, we will end up with 3 tables when considering 3NF as part of our design.

Many to Many relationships Example

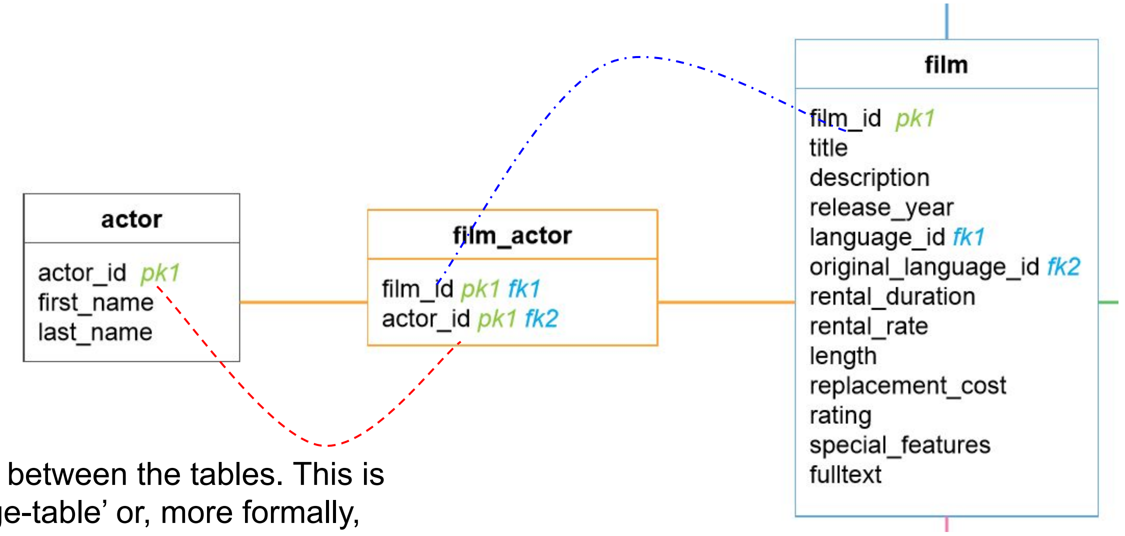
Consider the dvdstore example:

- An actor can be a cast member of several movies.
    - A movie can have several actors.

This is a “many to many” relationship.

Many to Many relationships Example

For this relationship to work we have defined two foreign keys in the film_actor table, the primary keys of each of the other two tables.Consequently we end up with three tables to describe this relationship:



Film_actor serves as a link between the tables. This is often referred to as a ‘bridge-table’ or, more formally, and associative entity

Creating Tables Example

We are now ready to evaluate the syntax for table creation and alteration. This is the Create table syntax for all 3 of the previous tables:

```
CREATE TABLE actor (
    actor_id serial NOT NULL,
    first_name varchar(45) NOT NULL,
    last_name varchar(45) NOT NULL,
    CONSTRAINT pk_actor_actor_id PRIMARY
KEY (actor_id)
);
```

In film_actor are actor_id and film_id foreign keys yet?

No!

```
CREATE TABLE film_actor (
    actor_id integer NOT NULL,
    film_id integer NOT NULL,
    CONSTRAINT pk_film_actor_actor_id_film_id PRIMARY KEY (actor_id, film_id)
);
```

```
CREATE TABLE film (
    film_id serial NOT NULL,
    title varchar(255) NOT NULL,
    description varchar(512),
    release_year smallint,
    language_id integer NOT NULL,
    original_language_id integer,
    rental_duration smallint DEFAULT 3 NOT NULL,
    rental_rate numeric(4,2) DEFAULT 4.99 NOT NULL,
    length smallint,
    replacement_cost numeric(5,2) DEFAULT 19.99 NOT NULL,
    rating varchar(5) DEFAULT 'G',
    CONSTRAINT pk_film_film_id PRIMARY KEY (film_id),
    CONSTRAINT ck_film_rating CHECK (rating IN ('G', 'PG', 'PG-13', 'R', 'NC-17'))
);
```

Creating Tables Example

We finish by specifying that actor_id and film_id are actually foreign keys. The DBMS does not assume this just because it has the same name, we must use the ALTER command:

```
ALTER TABLE film_actor
ADD FOREIGN KEY(film_id)
REFERENCES film(film_id);

ALTER TABLE film_actor
ADD FOREIGN KEY(actor_id)
REFERENCES actor(actor_id);
```