Database Design
&
Data DefinitionLanguage

Database Design

Normalization

Normalization is the process of organizing a database to reduce data redundancy and improve data integrity.

There are three (or more) **normal forms** that are typically followed to get to what is considered a good database design.

The **goal** is to get to third normal form (**3NF**) or beyond,

Why Normalize?

The main reasons for normalizing a database are:

- Minimize duplicate data
- Avoid data modification issues
- Simplify Queries

First Normal Form

In **1NF**, no table should contain duplicative columns that one could use to get other types of information. Every table should be organized in rows with primary keys that uniquely identify it.

The table below is unnormalized. It is not in first normal form because the Items field contains multiple items. How would we query the individual items for a customer?

Customer	Items
Dave Buster	Scissors, Tape, Glue
Molly McButter	8 x 10 Frame, Framing Mat
Dave Buster	Large Box

First Normal Form

Now we have a separate record for each customer and item. As it is now, we can identify each item by the customer and item. What happens if two customers buy the same item? What about if the same customer buys the same item twice though?

Customer	Item
Dave Buster	Scissors
Dave Buster	Tape
Dave Buster	Glue
Molly McButter	8 x 10 Frame
Molly McButter	Framing Mat
Dave Buster	Large Box

First Normal Form

Now we have a table in first normal form:

There are single items in the item column and each customer item record has a unique id.

Customer Item ID	Customer	Item
1	Dave Buster	Scissors
2	Dave Buster	Tape
3	Dave Buster	Glue
4	Molly McButter	8 x 10 Frame
5	Molly McButter	Framing Mat
6	Dave Buster	Large Box

Second Normal Form

In **2NF** the table must be in **1NF** and any non-key attribute must be dependent on the primary key.

In the table below, the primary key is a composite key made up of Course ID and Instructor ID. The table is not in second normal form because Course Name is dependent on Course ID but not Instructor ID. How would we resolve this?

Course ID	Instructor ID	Course Name
1	1	Java
2	3	.NET
1	2	Java

Second Normal Form

Now all the non-key data in the Course table relates to the primary key and we use a relator table to associate courses and instructors.

Course ID	Course Name
1	Java
2	.NET

Course ID	Instructor ID
1	1
2	3
1	2

Third Normal Form

In **3NF** the table must be in **2NF** and there must be no transitive functional dependency.

This means that if A is dependent on B and B is dependent on C then C is transitively dependent on A.

The table below is not in third normal form because Winner is dependent on the composite key of Category and Year and Winner DOB is dependent on Winner but NOT on Category and Year.

We hope Billie Eilish will forgive us for listing Childish Gambino as the 2020 Winner… it made for a better example.

Category	Year	Winner	Winner DOB
Best Solo Performance	2020	Lizzo	4/27/1988
Record Of The Year	2020	Childish Gambino	9/25/1983
Best Solo Performance	2019	Lady Gaga	3/28/1986
Record Of The Year	2019	Childish Gambino	9/25/1983

Third Normal Form

Now Winner is dependent on the composite key of Category and Year and Winner DOB is dependent on the Primary key in the Winner table. This is in 3NF.

Category	Year	Winner
Best Solo Performance	2020	Lizzo
Record Of The Year	2020	Childish Gambino
Best Solo Performance	2019	Lady Gaga
Record Of The Year	2019	Childish Gambino

We *still* hope Billie Eilish will forgive us for listing Childish Gambino as the 2020 Winner… it made for a better example.

Winner	Winner DOB
Lizzo	4/27/1988
Lady Gaga	3/28/1986
Childish Gambino	9/25/1983

Let's Design A Database!

Design Step 1 - Define Objects

- Start by defining your objects. Go through the requirements and decide what parts of the system should be your main objects. Don’t worry about how the objects relate to each other yet - just focus on what objects you will need.

- When your objects seem fairly complete, think about how they relate to each other. Some of your objects will be entities that are central to the problem you are trying to solve, while others will be used to support those entities. You may even decide some objects aren’t actually needed.

Design Step 2 - Identify Tables And Columns

- Major entities will usually translate into database tables. Each one will have attributes, which will become the table columns.

- You will need to decide how your entities will be broken into tables and what attribute columns each table will have. This is where some of the normalization techniques we discussed might be used. You will need to consider:
  - The information you will want to get from the database.
  - How that information will be used.

Design Step 3 - Define Tables

- Decide what column names and data types each table will have.

- Write the DDL script that will create the tables you have designed in the database.

Use draw.io to diagram your tables and relationships While Designing

Design Activity

Work with your group to come up with a good design for the data listed here.

We'll reconvene in about 15 minutes to review everyone's designs and decide on a final design.

Gallery Customer History Form

Customer Name

Jackson, Elizabeth
123 – 4th Avenue
Fonthill, ON
L3J 4S4

Phone (206) 284-6783

Purchases Made

Artist	Title	Purchase Date	Sales Price
03 - Carol Channing	Laugh with Teeth	09/17/2000	7000.00
15 - Dennis Frings	South toward Emerald Sea	05/11/2000	1800.00
03 - Carol Channing	At the Movies	02/14/2002	5550.00
15 - Dennis Frings	South toward Emerald Sea	07/15/2003	2200.00

The Gill Art Gallery wishes to maintain data on their customers, artists and paintings. They may have several paintings by each artist in the gallery at one time. Paintings may be bought and sold several times. In other words, the gallery may sell a painting, then buy it back at a later date and sell it to another customer.

What Did You Come Up With?

Data Definition Language (DDL)

Just like DML is used to manipulate and query data, DDL is used to create and define database structures.

- **DDL** consists of the set of commands used to **define** tables, keys, constraints, and other metadata that comprise a database **schema**.

- A **schema** is a skeleton structure of the database that focuses on the tables and constraints, not the data.

Data Definition Language (DDL)

Creating A Database:

```
CREATE DATABASE world;
```

Dropping A Database:

```
DROP DATABASE world;
```

Note that the `createdb` *(and* `dropdb` *for those who used it) command we used is a Postgres utility that wraps this functionality.*

Data Definition Language (DDL)

Creating Tables:

The **CREATE TABLE** statement **creates** a new **table** and **defines** its **structure**. The **columns** and **data types** are **required**. Primary Keys, Foreign Keys, and other **constraints** are typically included but can be added later.

Data Definition Language (DDL)

Creating Tables:

```
CREATE TABLE table_name (
        column_name_1   data_type(size) NOT NULL,
        column_name_2   data_type(size) ,
        column_name_3   data_type,
        CONSTRAINT pk_column_1 PRIMARY KEY (column_name_1),
        CONSTRAINT fk_column_2 FOREIGN KEY (coumn_name_2) REFERENCES table_name_2 (column_name)
);
```

- The column name is exact column name.
- The data type indicates what it holds.
- The size specifies the max length of the column, when required by the data type.

Data Definition Language (DDL)

CREATE TABLE Example:

```
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)
);
```

Data Definition Language (DDL)

Removing tables:

```
DROP TABLE table_name;
DROP TABLE IF EXISTS table_name;
```

Altering tables:

The ALTER TABLE command can change the structure of a table or add or delete constraints to the able.

```
ALTER TABLE table_name ADD CONSTRAINT pk_constraint_name
PRIMARY KEY (column_name);
```

Data Definition Language (DDL)

ALTER TABLE Examples:

```
ALTER TABLE campground ADD CONSTRAINT
pk_campground_campground_id PRIMARY KEY (campground_id);

ALTER TABLE site ADD FOREIGN KEY (campground_id)
REFERENCES campground(campground_id);

ALTER TABLE site ADD CONSTRAINT
chk_site_num CHECK(site_number > 0);
```

Sequences

Sequences can be use to **auto-generate sequential unique numbers**. They are often used to generate **ids** to be used as **surrogate keys**. The start at 0, unless otherwise specified, and increment the value each time a new value as needed.

Can be **created manually**:

```
CREATE SEQUENCE seq_name;
```

Or **when creating** a **table** by specifying a **column data type** of **serial**.

```
column_name   serial
```

Sequences

To get the next value from the sequence:

```
SELECT nextval(sequence_name);
```

Example:

```
SELECT nextval('customer_seq');
```

When a column is specified as type serial, the next value of the auto-created sequences will be used when inserting a record and omitting the field corresponding to the sequence.

Data Definition Language (DDL)

SEQUENCE Example:

```
CREATE SEQUENCE sample_seq;

SELECT nextval('sample_seq');
```

Database Control Language (DCL)

Database Control Language (DCL) is used to administer the database, users, and permission.

- Database commands are executed under the context of an **authenticated user**.
- Database control language commands are used to create users, drop users, grant privileges and revoke privileges.
- Important to know:
  - `GRANT`
  - `REVOKE`

Exercise Notes

Make sure to insert the data specified in the requirements as part of your SQL script.