**DATABASE DESIGN**

**Part 1**

**Normalisation**

Normalisation is the process of organising the columns of a relational database to reduce data redundancy and improve data integrity.

Before you begin normalising your database, you must define the role of the application being developed.

Keys

* Primary key: - always has a value which never changes. It is a unique value for each record in a table (e.g. national insurance number)
* Foreign key: - A key used to link 2 tables together. A foreign key is a field (or collection of fields) in one table that refers to the primary key in another table

\*If no logical primary key exists, invent one\*

*Example 1.1:*

A message board where users can post messages and other users can reply. Users will need to register and log in with an email/password combination, in order to post messages. I also expect that there could be multiple forums for different subjects.

|  |  |
| --- | --- |
| **Column** | **Example** |
| Message ID | 325 |
| Username | Rezaa91 |
| Password | Mypass |
| Actual name | Ali Issaee |
| User email | [Email@example.com](mailto:Email@example.com) |
| Forum | MySQL |
| Message subject | Question about normalisation |
| Message body | I have a question about… |
| Message date | December 13, 2017 12:20 AM |

\*Ideally, your primary key should always be an integer\*

Relationships

Database relationships refer to how the data in one table relates to data in another.

3 types:

1. One-to-one
2. One-to-many
3. Many-to-many (avoid – use multiple one-to-many relationships instead)

1 – If one item in table A refers to one item in table B – e.g. each US citizen has only one social security number, and each security number applies to only one US citizen

2 – if one item in table A can apply to multiple items in table B (most common) – e.g. The terms male & female will apply to many people, but each person can only be one or the other

3 – if multiple items in table A can apply to multiple items in table B – e.g. A book can be written by multiple authors and authors can write multiple books

First nominal form (1NF)

Normalising a database is the process of changing the databases structure according to several rules, called forms. Your database should adhere to each rule exactly, and forms must be followed in order.

Every table in a database must have the following 2 qualities in order to be in first nominal form:

1. Each column must contain only one value (AKA atomic)
2. No table can have repeated groups of related data

\*Any columns that are no atomic should be broken in to multiple columns\*

\*If a table has repeating similar columns, then those should be turned into their own separate table\*

In example 1.1 – *actual name* is NOT 1NF compliant – you should create first name and last name columns.

Example 1.2 – showing point 2 from 1NF:

*Movies table*

|  |  |
| --- | --- |
| **Column** | **Example** |
| Movie id | 976 |
| Movie title | Star wars |
| Year released | 1955 |
| \*Director | George Lucas |
| \*Actor 1 | Ewan McGregor |
| \*Actor 2 | Liam Neeson |

\*Repeated and not atomic

In order to correct this:

*Movies-actor table*

|  |  |
| --- | --- |
| **Column** | **Example** |
| ID | 1 |
| Movie | Star wars |
| Actor first\_name | Ewan |
| Actor last\_name | McGregor |

*However, this violates the 2NF rule*

Second nominal form (2NF)

For a database to be in a 2NF, the database must first already be in 1NF. Then, every column in the table that is not a key (i.e. foreign key), must be dependent upon the primary key.

You can normally identify a column that violates this rule when it has non-key values that are the same in multiple rows. Such values should be stored in their own table and related back to the original table through a key.

Example 1.2 would violate the 2NF rule as the movies table could have a director listed multiple times – the fix is to create a separate directors table that stores the directors info and assigns each director a primary key. To tie the director back to the movies, the directors primary key would also be a foreign key in the movies table.

To correct example 1.2 you would need 4 tables:

*\*Directors to movies is a one-to-many relationship, movies to movies-actors is a one-to-many relationship, movies-actors to actors is a many-to-one relationship*

To make a database 2NF compliant:

1. Identify any non-key columns that aren’t dependent upon the tables primary key
2. Create new tables accordingly
3. Assign or create new primary keys
4. Create the requisite foreign keys and indicate the relationships

For example 1.1, username, first\_name, last\_name, email and forum values are all non-keys and none are dependent on the message\_id…

To correct:

*\*users to messages is one-to-many relationship, messages to forums is a many-to-one relationship*

Third nominal form (3NF)

A database is 3NF if it is in 2NF and every non-key column is mutually independent. You would know if you have a 3NF violation if changing the value in one column would require changing the value in another.

To make a database 3NF compliant:

1. Identify any fields in any tables that are interdependent
2. Create new tables accordingly
3. Assign or create new primary keys
4. Create the requisite foreign keys that link any of the relationships

\*after walking through the normalisation process, it is best to review the design one more time\*

**Part 2**

**Creating indexes**

Indexes are a special system that databases use to improve the performance of SELECT queries. Indexes can be placed on one or more columns, of any data type, effectively telling MySQL to pay particular attention to those values.

Indexes are best used on columns:

* That are frequently used in the WHERE part of a query
* That are frequently used in an ORDER BY part of a query
* That are frequently used as the focal point of a JOIN

You should not index columns that:

* Allow for NULL values
* Have a very limited range of values (Y/N or 1/0)

4 types of indexes:

|  |  |
| --- | --- |
| INDEX | Standard |
| UNIQUE | Requires each row to have a unique value for that column |
| FULLTEXT | For performing full text searches |
| PRIMARY KEY | A particular UNIQUE index |

\*A column should only ever have a single index on it\*

To establish an index when creating a table, this clause is added to the CREATE TABLE command:

INDEX\_TYPE index\_name(columns)

INDEX full\_name(last\_name, first\_name)

To create indexes:

1. Add a PRIMARY KEY index on all primary keys
2. Add UNIQUE indexes to any columns whose value cannot be duplicated within the table e.g. username, email…
3. Add FULLTEXT if appropriate
4. Add standard indexes to columns frequently used in WHERE, ORDER BY clauses and JOIN

Example:

|  |  |  |
| --- | --- | --- |
| **Column\_name** | **Table** | **Index\_type** |
| Forum\_id | Forums | Primary |
| Name | Forums | Unique |
| Message\_id | Messages | Primary |
| Forum\_id | Messages | Index |
| Parent\_id | Messages | Index |
| User\_id | Messages | Index |
| Date\_entered | Messages | Index |
| User\_id | Users | Primary |
| Username | Users | Unique |
| Password | Users | Index |
| Email | Users | Unique |

\*indexes can be created after you already have a populated table\*

**Using different table types**

AKA storage engine. Two common engines = MyISAM & InnoDB (default engine).

SHOW ENGINES – will reveal available storage engines and the default engine

To specifiy storage engine:

CREATE TABLE tablename(

Column\_name COLUMN\_TYPE,

Column\_name COLUMN\_TYPE…

)ENGINE = type;

\*if any of your tables require a FULLTEXT index, make it a MyISAM table as not supported by InnoDB\*

\*if any of your tables require support for transactions make it an InnoDB table\*