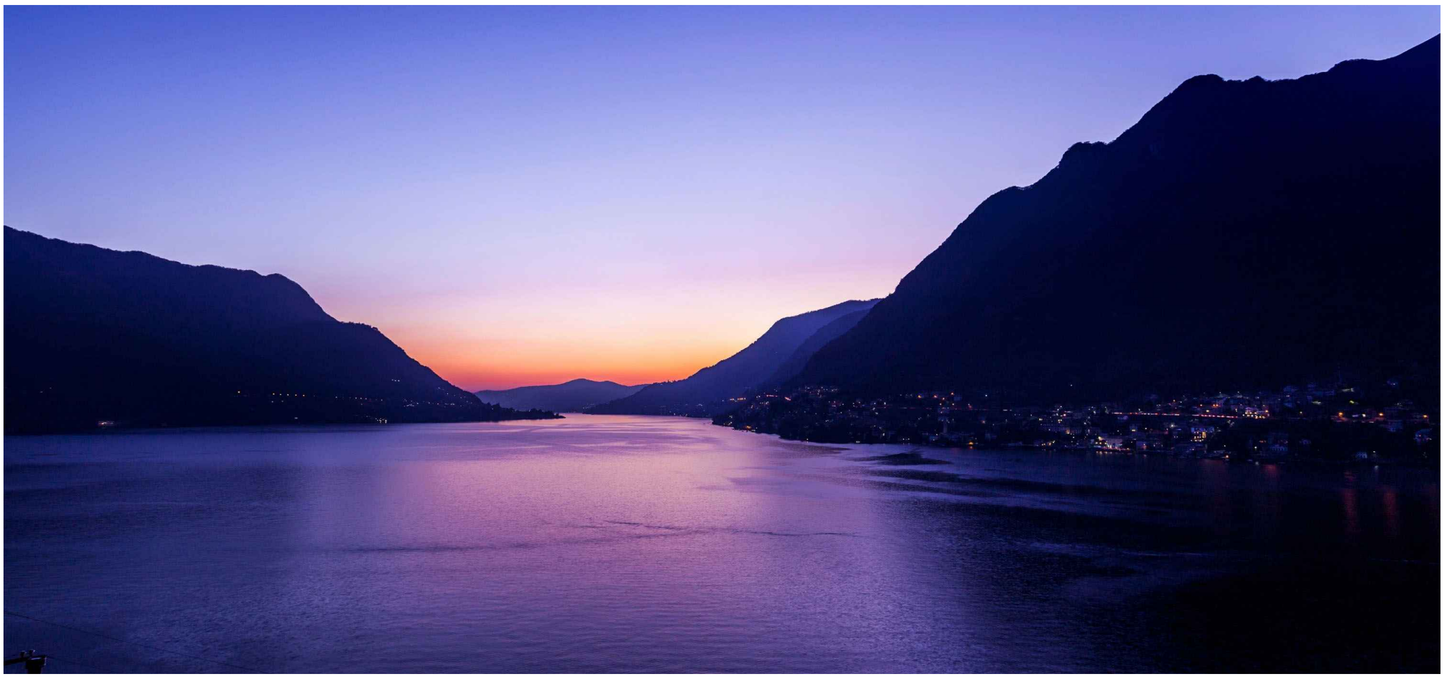


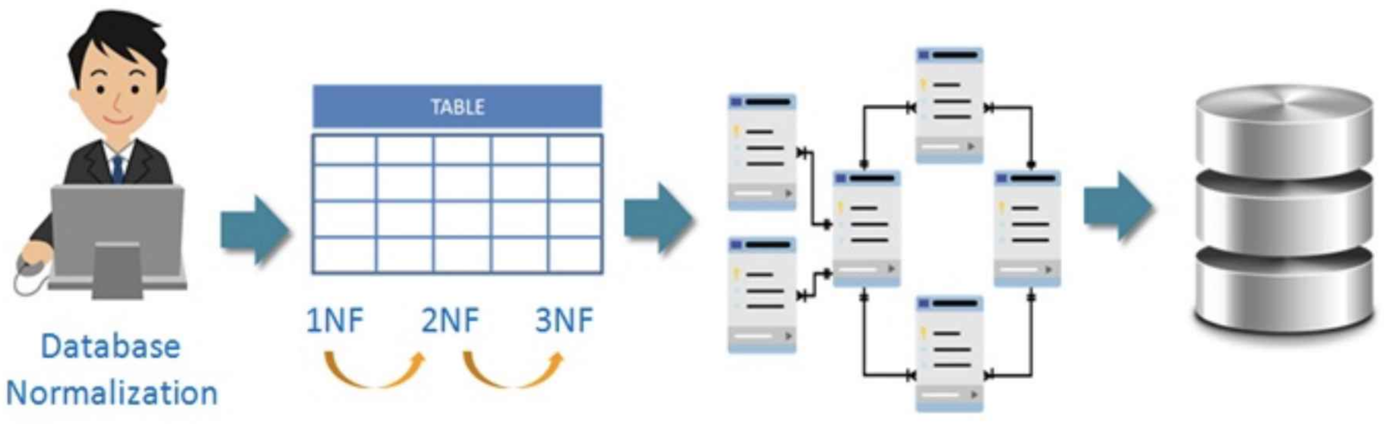
11-Database Normalization

Author: Vincent Lau

Learning Objectives

- Understand the purpose of database normalization
- Understand the advantages and drawbacks of database design with normalization

Introduction



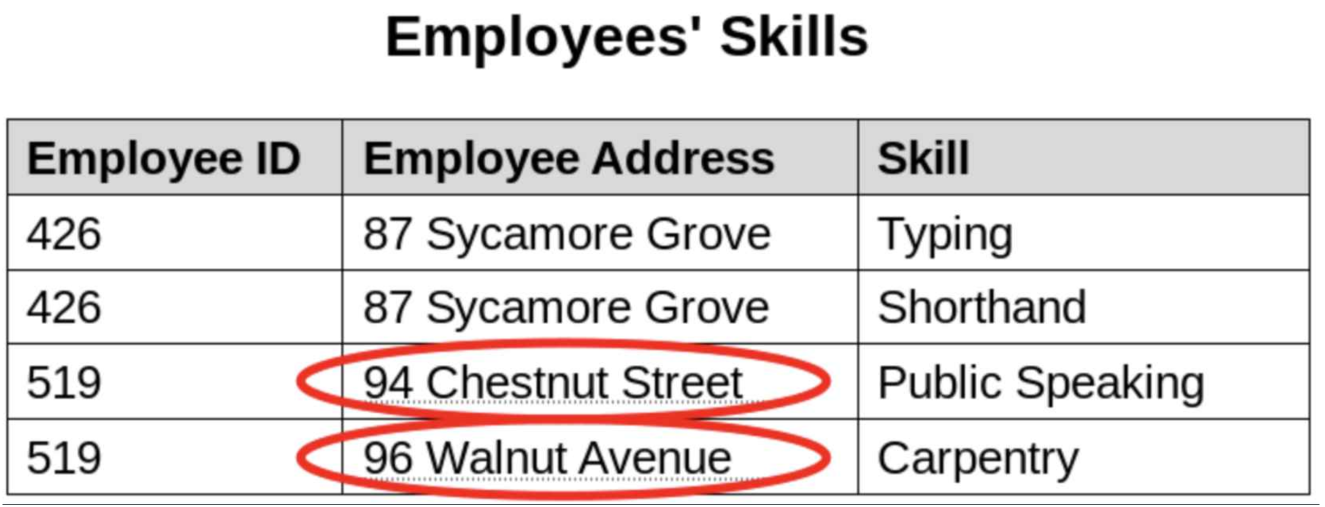
Database normalization is a process in relational database design that aims to minimize data redundancy and improve data integrity by organizing data into separate tables and defining relationships between them.

The goal is to eliminate anomalies and inconsistencies that can arise when data is stored redundantly in multiple places. The normalization process involves breaking down larger tables into smaller, related tables based on certain rules or normal forms.

Problem without Normalization

1 - Update Anomaly

- When the same data is stored in more than one rows. A change of data may need to be applied to multiple records. The relation will be left in an inconsistent state if updates are being made to some data but not others.
- **Example:** The address for employee ID 519 should be same. What if we update them inconsistently? How to avoid it?



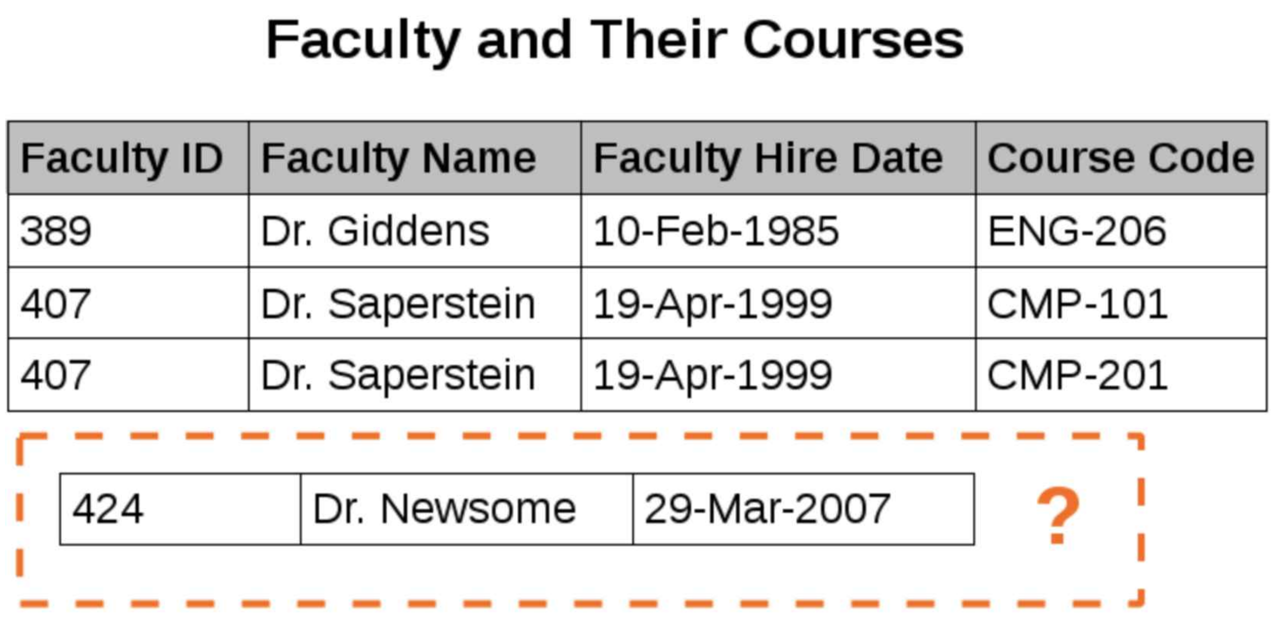
Employees' Skills

Employee ID	Employee Address	Skill
426	87 Sycamore Grove	Typing
426	87 Sycamore Grove	Shorthand
519	94 Chestnut Street	Public Speaking
519	96 Walnut Avenue	Carpentry

2 - Insertion Anomaly

- When we try to insert a record into a table, we may end up adding incomplete data because some irrelevant data has not been known yet.

- **As a result, we have to wait for the value of "Course Code" ready before inserting the records in the table.**
- For example, "ENG-206" actually represents a course, but the course code can be changed due to many reasons. So, inserting a record into the table has to wait for the "course code" finalized if we DO NOT normalize the table.
- If we normalize the table into 2 tables (FACULTY & FACULTY_COURSE), then we can update the course_code to FACULTY_COURSE later on.



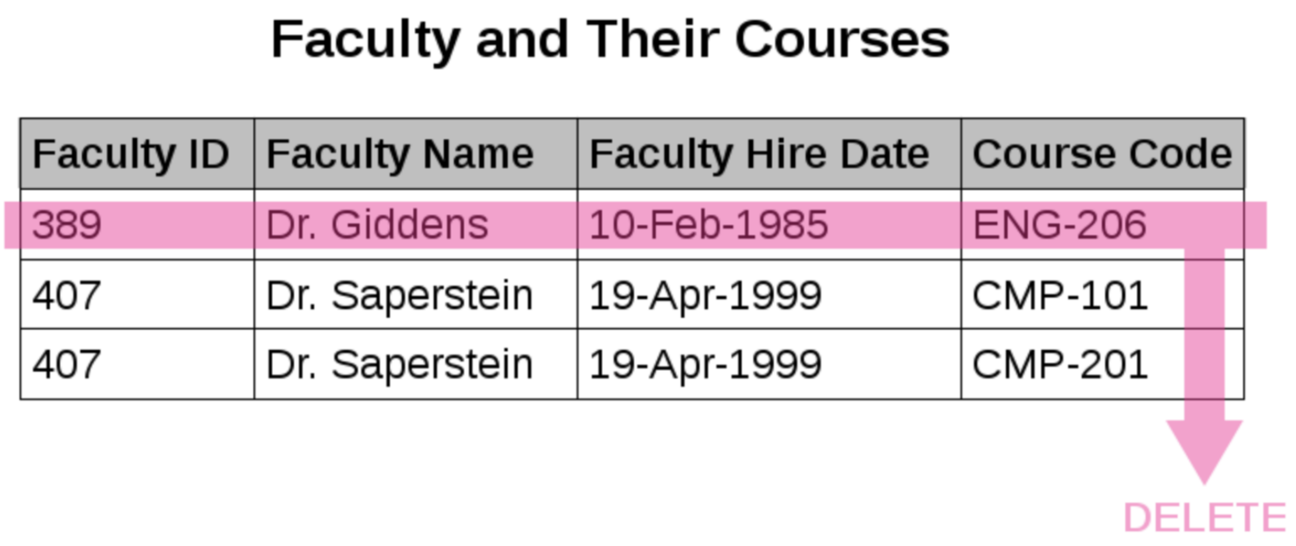
Faculty and Their Courses

Faculty ID	Faculty Name	Faculty Hire Date	Course Code
389	Dr. Giddens	10-Feb-1985	ENG-206
407	Dr. Saperstein	19-Apr-1999	CMP-101
407	Dr. Saperstein	19-Apr-1999	CMP-201

424	Dr. Newsome	29-Mar-2007	?

3 - Delete Anomaly

- Data may be **erroneously retained** or **mistakenly deleted**.
- Similar problem with "Insert Anomaly". You cannot partially delete the record if needed.
- **For example, in table "Faculty", if you want to remove a "Course", what you can do is just update the the "Course Code" to NULL, but not delete the record.**



Faculty and Their Courses

Faculty ID	Faculty Name	Faculty Hire Date	Course Code
389	Dr. Giddens	10-Feb-1985	ENG-206
407	Dr. Saperstein	19-Apr-1999	CMP-101
407	Dr. Saperstein	19-Apr-1999	CMP-201

```
-- Solution: (Many to Many)
```

```sql
CREATE TABLE FACULTY (
    ID INTEGER NOT NULL AUTO_INCREMENT,
    FACULTY_NAME VARCHAR(100) NOT NULL,
    FACULTY_HIRE_DATE DATE NOT NULL,
    PRIMARY KEY(ID)
);

CREATE TABLE COURSE (
    ID VARCHAR(3) NOT NULL AUTO_INCREMENT,
    COURSE_CODE VARCHAR(7) NOT NULL, -- "ENG-206", "CMP-101"
    -- FACULTY_ID INTEGER NOT NULL,
    PRIMARY KEY(ID),
    -- FOREIGN KEY(FACULTY_ID) REFERENCES FACULTY(ID)
);

CREATE TABLE FACULTY_COURSE (
    ID INTEGER NOT NULL AUTO_INCREMENT,
    COURSE_ID VARCHAR(3) NOT NULL,
    FACULTY_ID INTEGER NOT NULL,
    PRIMARY KEY(ID)
    FOREIGN KEY(COURSE_ID) REFERENCES COURSE(ID),
    FOREIGN KEY(FACULTY_ID) REFERENCES FACULTY(ID)
);

-- Solution: (One to Many - One Course is owned by one Faculty only)
CREATE TABLE FACULTY (
    ID INTEGER NOT NULL AUTO_INCREMENT,
    FACULTY_NAME VARCHAR(100) NOT NULL,
    FACULTY_HIRE_DATE DATE NOT NULL,
    PRIMARY KEY(ID)
);

CREATE TABLE COURSE (
    ID VARCHAR(3) NOT NULL AUTO_INCREMENT,
    COURSE_CODE VARCHAR(7) UNIQUE NOT NULL, -- "ENG-206", "CMP-101"
    FACULTY_ID INTEGER NOT NULL,
    PRIMARY KEY(ID),
    FOREIGN KEY(FACULTY_ID) REFERENCES FACULTY(ID)
);
```

Database normalization - Wikipedia

Implement Normalization

`BCNF` stands for Boyce-Codd Normal Form, and it's a higher level of database normalization that builds upon the concepts of 1st, 2nd, and 3rd Normal Forms. BCNF is used to further eliminate redundancy and anomalies in a relational database by ensuring that every determinant (attribute that determines the value of another attribute) is a superkey.

A table is in BCNF if, for every non-trivial functional dependency X -> Y, X is a superkey. This means that all attributes of a table should be functionally dependent only on the entire primary key, and not on any subset of it.

There are several normal forms, each building on the previous one, with the ultimate goal of achieving a high level of data integrity and reducing redundancy. The most common normal forms are:

1. First Normal Form (1NF)

Eliminates repeating groups and ensures that each column contains atomic (indivisible) values.

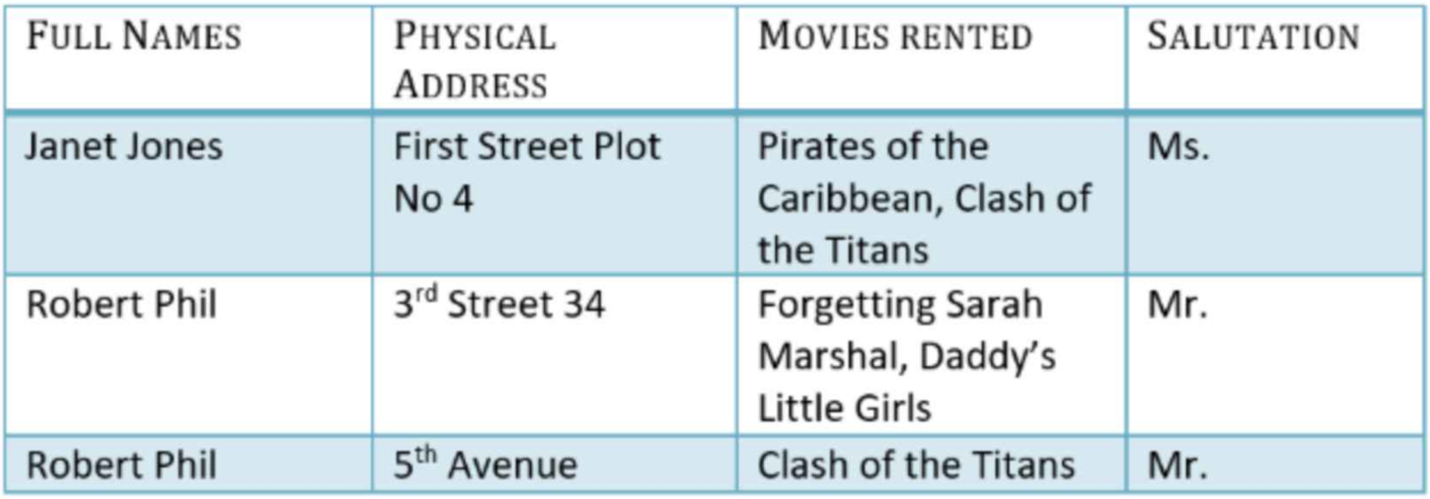
2. Second Normal Form (2NF)

Eliminates partial dependencies by ensuring that non-key attributes are fully functionally dependent on the entire primary key.

3. Third Normal Form (3NF)

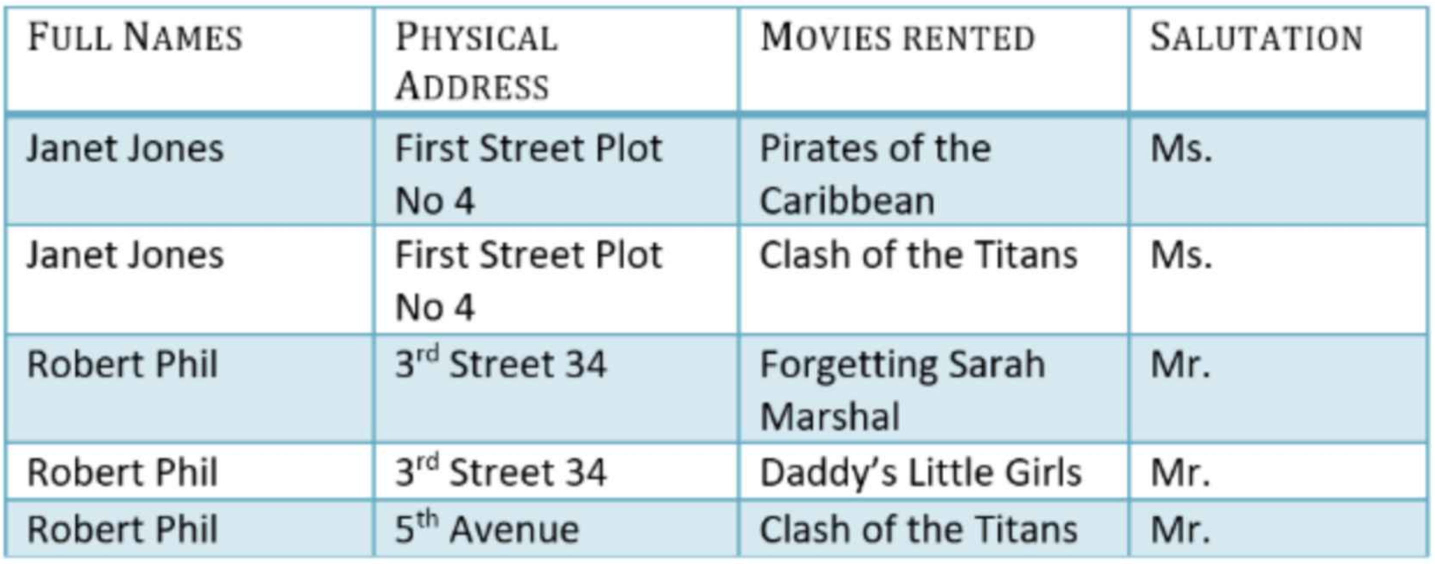
Eliminates transitive dependencies by ensuring that non-key attributes are not dependent on other non-key attributes.

Example of Normalization



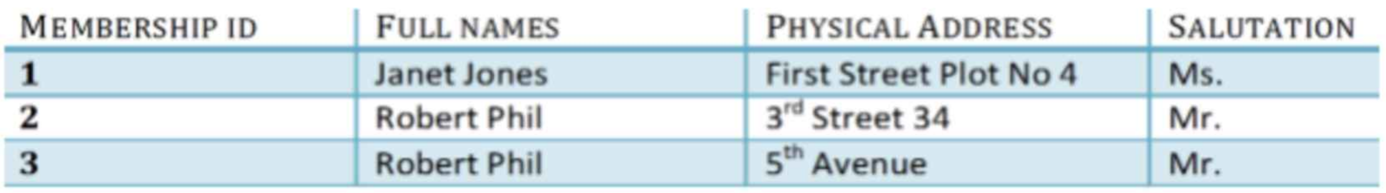
FULL NAMES	PHYSICAL ADDRESS	MOVIES RENTED	SALUTATION
Janet Jones	First Street Plot No 4	Pirates of the Caribbean, Clash of the Titans	Ms.
Robert Phil	3rd Street 34	Forgetting Sarah Marshal, Daddy's Little Girls	Mr.
Robert Phil	5th Avenue	Clash of the Titans	Mr.

- The above column "**MOVIES RENTED**" **violates the 1NF rule**.
- **An attribute (column) of a table cannot hold multiple values.** It should hold only `atomic values`. Each row should be able to be *`uniquely`* identified by either a column or a set of columns. (1NF)

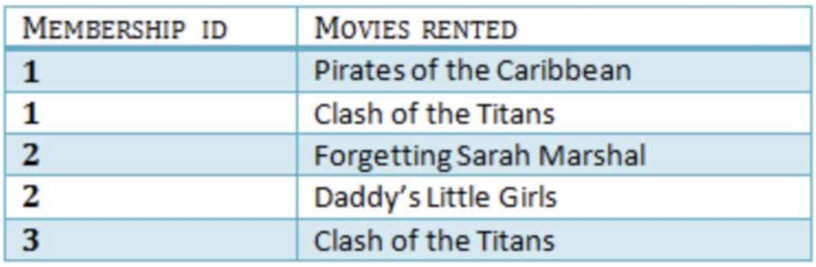


FULL NAMES	PHYSICAL ADDRESS	MOVIES RENTED	SALUTATION
Janet Jones	First Street Plot No 4	Pirates of the Caribbean	Ms.
Janet Jones	First Street Plot No 4	Clash of the Titans	Ms.
Robert Phil	3rd Street 34	Forgetting Sarah Marshal	Mr.
Robert Phil	3rd Street 34	Daddy's Little Girls	Mr.
Robert Phil	5th Avenue	Clash of the Titans	Mr.

- No `partial dependency` - Remove subsets of data that apply to multiple rows of a table and place them in separate tables. Create relationships between these new tables and their predecessors through the use of ***foreign keys.***
    - `Partial dependency` means a non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.



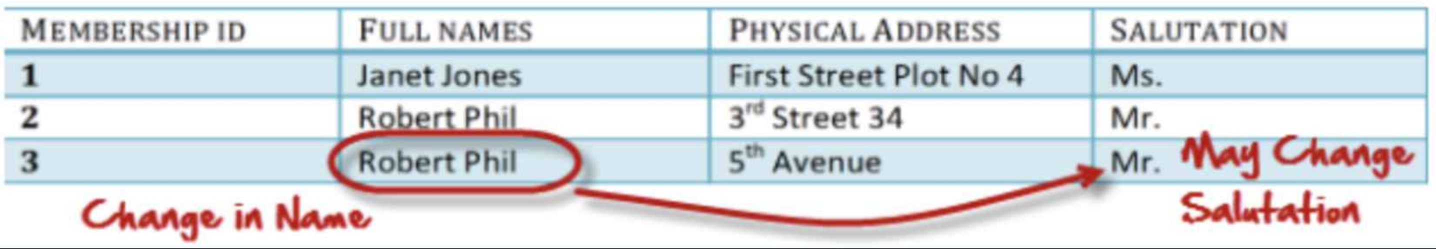
MEMBERSHIP ID	FULL NAMES	PHYSICAL ADDRESS	SALUTATION
1	Janet Jones	First Street Plot No 4	Ms.
2	Robert Phil	3rd Street 34	Mr.
3	Robert Phil	5th Avenue	Mr.



MEMBERSHIP ID	MOVIES RENTED
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans

- No `transitive dependency` - Remove columns that are not dependent upon the primary key.

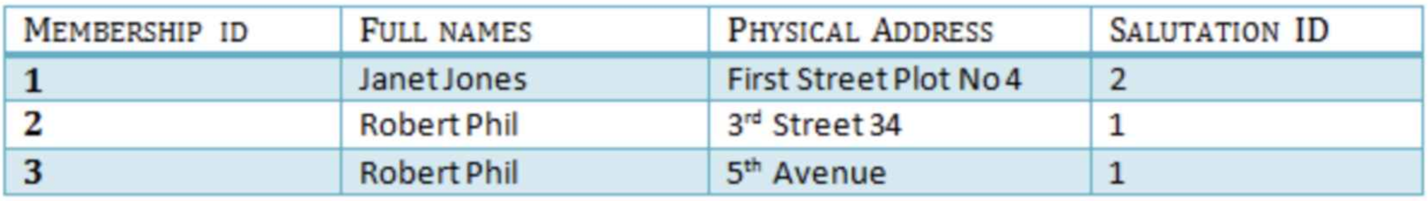
Problem



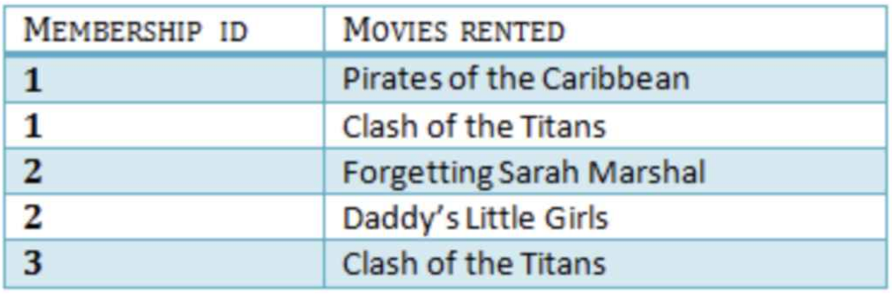
MEMBERSHIP ID	FULL NAMES	PHYSICAL ADDRESS	SALUTATION
1	Janet Jones	First Street Plot No 4	Ms.
2	Robert Phil	3rd Street 34	Mr.
3	Robert Phil	5th Avenue	Mr.

May Change Salutation
Change in Name

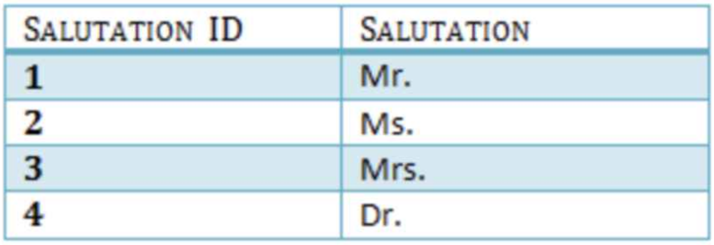
Solution



Membership ID	Full Names	Physical Address	Salutation ID
1	Janet Jones	First Street Plot No 4	2
2	Robert Phil	3rd Street 34	1
3	Robert Phil	5th Avenue	1



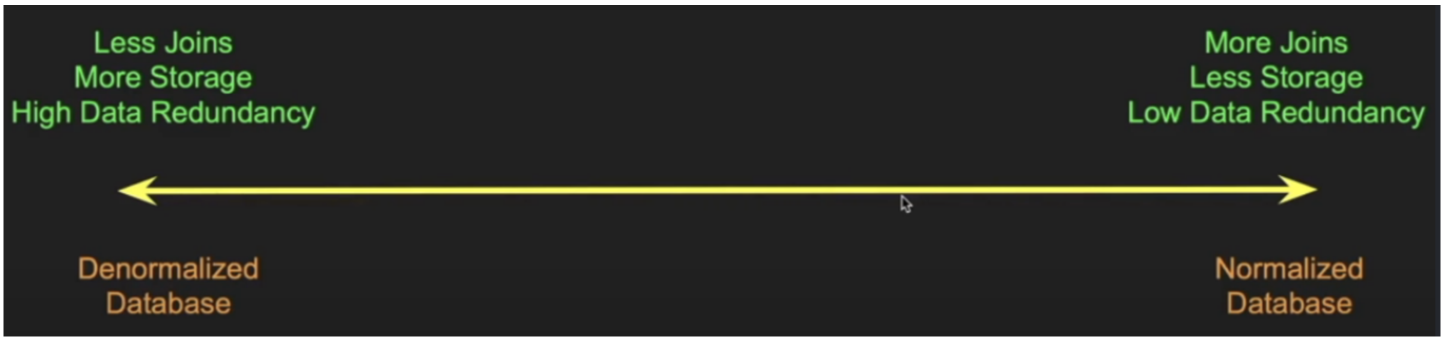
Membership ID	Movies rented
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans



Salutation ID	Salutation
1	Mr.
2	Ms.
3	Mrs.
4	Dr.

- For **PHYSICAL ADDRESS** in the first table, it can be further separated into a table with additional fields such as BUILDING, STREET, CITY and COUNTRY, where each row represents a physical address and is identified by a unique ADDRESS_ID. A junction table is then created to map each MEMBERSHIP_ID to an ADDRESS_ID, where additional fields such as IS_ACTIVE and MODIFIED_DATE can be included.
- For each `functional dependency` X -> Y, X should be the **super key** of the table.
  - A `functional dependency` is a constraint that specifies the relationship between two sets of attributes where one set can accurately determine the value of other sets. For example, MEMBERSHIP ID -> FULL NAMES, SALUTATION ID -> SALUTATION, and MEMBERSHIP ID -> MOVIES RENTED.

Why Normalization



Advantanges

Data Integrity

Normalization helps to maintain accurate and consistent data by reducing redundancy and the chances of inconsistencies.

Efficient Updates

Since data is stored in fewer places, updates and modifications are easier and require fewer changes.

Reduced Redundancy

Data is stored only once, which saves storage space and helps in maintaining a more compact database.

Easier Maintenance

Changes to the database schema are less complex, leading to simpler and more manageable database maintenance.

Improved Query Performance

Normalization can lead to more efficient query execution due to smaller tables and well-defined relationships.

Downsides

Increased Complexity

Highly normalized databases can lead to complex joins and queries, which might negatively impact query performance.

Additional Joins

While normalization reduces redundancy, it can increase the number of joins needed to retrieve related data, impacting query performance.

Design Trade-offs

Striking the right balance between normalization and denormalization requires careful consideration **based on usage patterns.**

Performance Trade-offs

In some cases, excessive normalization might lead to performance issues due to increased join complexity.

Maintaining Relationships

Maintaining relationships and enforcing constraints across multiple normalized tables can sometimes become complex.

It's important to note that normalization is not always the best approach for every situation. The level of normalization should be chosen based on the specific needs of your application, considering factors like query patterns, performance requirements, and ease of maintenance. In some cases, controlled denormalization (introducing some redundancy) might be appropriate to optimize for certain query scenarios.