# SCHEMA REFINEMENT and NORMAL FORMS

Session 1:INTRODUCTION TO SCHEMA REFINEMENT

## Introduction to Schema Refinement

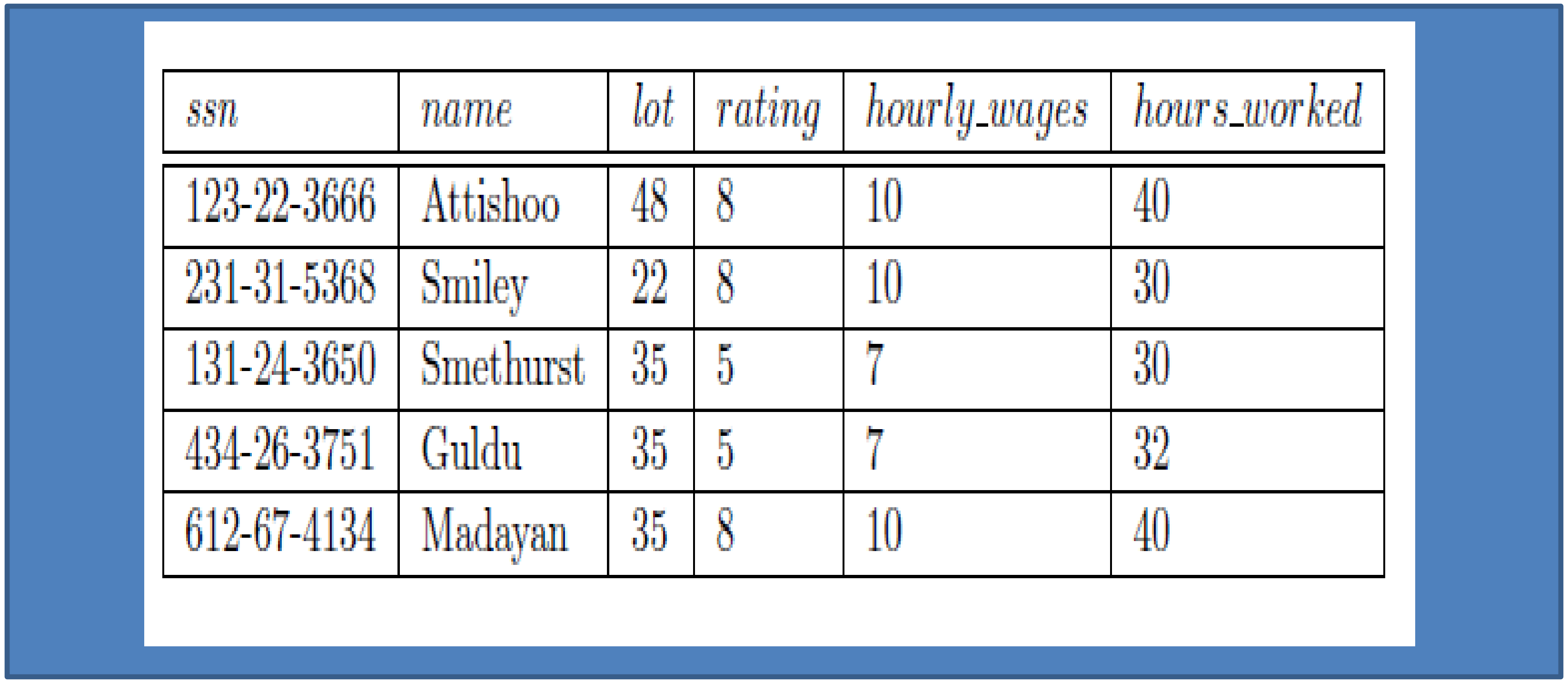
- Schema refinement, also known as normalization, is a critical process in database design aimed at improving the structure of a database schema.
- The main objective is to eliminate redundancy and avoid potential anomalies during data manipulation.
- This process involves decomposing larger tables into smaller, more manageable ones while ensuring that the original data can still be reconstructed without loss or inconsistency.

## Problems Caused by Redundancy:

- **Redundant Storage:** Redundancy occurs when the same piece of data is stored in multiple places within a database. For example, consider a table where the same hourly wage value is stored multiple times for employees with the same rating. This redundancy can lead to inefficient use of storage space.
- **Update Anomalies:** When data is duplicated, updating one instance of the data without updating all instances can lead to inconsistencies. For example, if the hourly wage for a particular rating is updated in one row but not in others, the database will contain conflicting information.
- **Insertion Anomalies**: Insertion anomalies occur when certain data cannot be inserted into the database without the presence of other data. For instance, if a table requires both an employee's information and their wage, it may be impossible to add a new employee without knowing their wage, even if that information is not yet available.
- **Deletion Anomalies:** Deletion anomalies occur when the removal of data also unintentionally removes other important information. For instance, deleting an employee record might also delete the only reference to a particular hourly wage, thereby losing the association between a wage and a rating.

# Example of Redundancy

- Consider a table `Hourly_Emps` with the following structure:
  - In this table, the `ssn` is the primary key, and it uniquely identifies each employee. There's a functional dependency where the `hourly wages` depend on the `rating`. This means that for a given rating, there is a specific hourly wage.



| *ssn* | *name* | *lot* | *rating* | *hourly_wages* | *hours_worked* |
|---|---|---|---|---|---|
| 123-22-3666 | Attishoo | 48 | 8 | 10 | 40 |
| 231-31-5368 | Smiley | 22 | 8 | 10 | 30 |
| 131-24-3650 | Smethurst | 35 | 5 | 7 | 30 |
| 434-26-3751 | Guldu | 35 | 5 | 7 | 32 |
| 612-67-4134 | Madayan | 35 | 8 | 10 | 40 |

-

## Problems Highlighted by Example:

- - The same `rating` might appear in multiple rows with the same `hourly wages`, leading to redundant data.
- - Updating the hourly wage for a specific rating would require updates to multiple rows, creating the risk of inconsistencies.
- - It may be impossible to insert a new employee without knowing their wage.
- - Deleting an employee who is the last one with a particular rating would remove the only reference to that rating's wage.

## Use of Decompositions

- Decomposition is the process of breaking down a larger table into smaller tables to eliminate redundancy.
- The goal is to remove redundancy while preserving the ability to reconstruct the original table using joins.

## Use of Decompositions

- For example, to solve the redundancy in the `Hourly_Emps` table, it can be decomposed into two smaller tables:
- 1. Hourly_Emps2 (`ssn`, `name`, `lot`, `rating`, `hours worked`): This table captures the essential employee details without the hourly wage.
- 2. Wages (`rating`, `hourly wages`): This table captures the relationship between the rating and the hourly wage.

# Use of Decompositions

| *ssn* | *name* | *lot* | *rating* | *hours_worked* |
|---|---|---|---|---|
| 123-22-3666 | Attishoo | 48 | 8 | 40 |
| 231-31-5368 | Smiley | 22 | 8 | 30 |
| 131-24-3650 | Smethurst | 35 | 5 | 30 |
| 434-26-3751 | Guldu | 35 | 5 | 32 |
| 612-67-4134 | Madayan | 35 | 8 | 40 |

| *rating* | *hourly_wages* |
|---|---|
| 8 | 10 |
| 5 | 7 |

## Use of Decompositions

- This decomposition eliminates redundancy.
  - The `Hourly_Emps2` table does not store wages, and any updates to the wage can be made in the `Wages` table without inconsistency.
  - Additionally, it resolves insertion and deletion anomalies because the wage for a rating can be maintained independently of the employees.

## Problems Related to Decomposition

- While decomposition can solve many issues, it can also introduce new problems, particularly if not done carefully.
- Two important questions must be asked repeatedly:
  - 1. Do we need to decompose a relation?
  - What problems (if any) does a given decomposition cause?

## Problems Related to Decomposition

- **To help with the first question, several *normal forms* have been proposed for relations.**
  - If a relation schema is in one of these normal forms, we know that certain kinds of problems cannot arise.
  - Considering the normal form of a given relation schema can help us to decide whether or not to decompose it further.
  - If we decide that a relation schema must be decomposed further, we must choose a particular decomposition (i.e., a particular collection of smaller relations to replace the given relation).

# Problems Related to Decomposition

- **With respect to the 2nd question, two properties of decomposition must be understood:**
- - Lossless-Join Property: This property ensures that the original table can be reconstructed by joining the decomposed tables without loss of information. A decomposition that lacks this property can lead to data loss or incorrect data retrieval.
- - Dependency Preservation: This property ensures that all functional dependencies in the original table are still enforceable after decomposition. If this property is not preserved, it may become necessary to join tables to enforce constraints, which can be inefficient.

## Drawback of decomposition:

- A serious drawback of decompositions is that queries over the original relation may require us to join the decomposed relations.
- It can also introduce the need for more complex queries that involve joining tables which can impact performance.
- In some cases, it may be better to accept some level of redundancy rather than decompose the table, depending on the specific needs of the application and the nature of the queries being performed.
- A good database designer should have a firm grasp of normal forms and what problems they (do or do not) alleviate, the technique of decomposition, and potential problems with decompositions.