# CSCI 4333 Section 1 Design of DB Systems

## 4/8/2024 (self - annotation)

**Introudction to Database Design**

K. Yue

**1. Introduction**

* Bad database/table designs result in unnecessary redundancy: redundancy with little or no benefit.
* However, redundancy can serve many purposes, and are employed frequently in computer science.
* Problems:
  1. Inefficient storage
  2. Anomaly: data inconsistency, loss of data integrity, difficulties in maintenance.

***Example:***

Consider the vastly simplified and poorly-designed relation/table:

Employee(EmpId, Name, DeptId, ManagerId).

Assumptions made:

1. Every employee works for only one department.
2. Every department has an unique DeptId.
3. Every employee is represented as a tuple in the Employee relation.
4. Every employee has an unique EmpId.
5. Every department has one manager, identified by the ManagerId

Thus, EmpId is a candidate key (CK). An instance of Employee:

|  |  |  |  |
| --- | --- | --- | --- |
| **EmpId** | **Name** | **DeptId** | **ManagerId** |
| 101 | Lady Gaga | *D123* | *M10* |
| 122 | Brad Pitts | *D123* | *M10* |
| 140 | Lebron James | *D123* | *M10* |
| 155 | Narendra Modi | D222 | M21 |
| 167 | Jennifer Lopez | D222 | M21 |
| 311 | John Smiths | D300 | M33 |

**Problem:**

* Unnecessary redundancy: e.g., the fact that manager M10 manages Department D123 is stored in three rows.

**1.1 Update Anomaly:**

(a) M44 becomes the new manager of Department D123

* Inefficiency in update: needs to update all three rows in this example.
* Potential (logical data) inconsistency.

|  |  |  |  |
| --- | --- | --- | --- |
| **EmpId** | **Name** | **DeptId** | **ManagerId** |
| 101 | Lady Gaga | D123 | *M10 -> M44* |
| 122 | Brad Pitts | D123 | *M10 -> M44* |
| 140 | Lebron James | D123 | *M10 -> M44* |
| 155 | Narendra Modi | D222 | M21 |
| 167 | Jennifer Lopez | D222 | M21 |
| 311 | John Smiths | D300 | M33 |

SQL solution: it works but is not efficient.

UPDATE Employee  
SET ManagerId = 'M44'  
WHERE DeptId = 'D123';

(b) Jennifer Lopez is reassigned to work for Department D300:

* Need to know the manager of D300.
* Potential inconsistency.

The table may become:

|  |  |  |  |
| --- | --- | --- | --- |
| **EmpId** | **Name** | **DeptId** | **ManagerId** |
| 101 | Lady Gaga | D123 | M10 |
| 122 | Brad Pitts | D123 | M10 |
| 140 | Lebron James | D123 | M10 |
| 155 | Narendra Modi | D222 | M21 |
| 167 | Jennifer Lopez | **D300** | M21 *(may not be updated to M33)* |
| 311 | John Smiths | D300 | M33 |

The intuitive SQL command:

UPDATE Employee  
SET DeptId = 'D300'  
WHERE Name = 'Jennifer Lopez';

will produce inconsistent result, as shown in the table above.

One needs to update both DeptId and ManagerId. However,

UPDATE Employee  
SET DeptId = 'D300',  
     ManagerId = (SELECT DISTINCT ManagerId FROM Employee WHERE DeptId = 'D300')  
WHERE Name = 'Jennifer Lopez';

will not work in MySQL as one cannot include a SELECT clause on the same table in the SET clause of an UPDATE statement.

A possible solution using a session variable, @M\_NO:

SELECT DISTINCT ManagerId INTO @M\_NO  
FROM Employee WHERE DeptId = 'D300';

UPDATE Employee  
SET DeptId = 'D300',  
     ManagerId = @M\_NO  
WHERE Name = 'Jennifer Lopez';

**1.2 Insertion Anomaly:**

It is not possible creating a new Department D777, with manager M40 but no employee working for it yet. This is because, as the PK, EmpId cannot be null.

|  |  |  |  |
| --- | --- | --- | --- |
| **EmpId** | **Name** | **DeptId** | **ManagerId** |
| 101 | Lady Gaga | D123 | M10 |
| 122 | Brad Pitts | D123 | M10 |
| 140 | Lebron James | D123 | M10 |
| 155 | Narendra Modi | D222 | M21 |
| 167 | Jennifer Lopez | D222 | M21 |
| 311 | John Smiths | D300 | M33 |
| ***????*** | **????** | **D777** | **M40 *(this row cannot be added)*** |

**1.3 Deletion Anomaly**

John Smiths no longer works here. Result: the information that M33 is the manager of Department D300 is also lost.

|  |  |  |  |
| --- | --- | --- | --- |
| **EmpId** | **Name** | **DeptId** | **ManagerId** |
| 101 | Lady Gaga | D123 | M10 |
| 122 | Brad Pitts | D123 | M10 |
| 140 | Lebron James | D123 | M10 |
| 155 | Narendra Modi | D222 | M21 |
| 167 | Jennifer Lopez | D222 | M21 |
| *~~311~~* | *~~John Smiths~~* | *~~D300~~* | *~~M33~~* |

**1.4 Decomposition**

A standard way of resolving unnecessary redundancy is by proper **decomposition**: breaking down a relation into two or more component relations.

***Example:*** the decomposition of the relation Employee into two relations:

1. Emp(EmpId, Name, DeptId)

|  |  |  |
| --- | --- | --- |
| **EmpId** | **Name** | **DeptId** |
| 101 | Lady Gaga | D123 |
| 122 | Brad Pitts | D123 |
| 140 | Lebron James | D123 |
| 155 | Narendra Modi | D222 |
| 167 | Jennifer Lopez | D222 |
| 311 | John Smiths | D300 |

2. Department(DeptId, ManagerId)

|  |  |
| --- | --- |
| **DeptId** | **ManagerId** |
| D123 | M10 |
| D222 | M21 |
| D300 | M33 |

To obtain the original relation Employee(EmpId, Name, DeptId, ManagerId) from

Emp(EmpId, Name, DeptId)  
Department(DeptId, ManagerId)

Relational algebra: using natural join, |x|.

Employee = Emp |x| Department

This decomposition is said to be a *lossless* decomposition.

SQL:

SELECT Emp.\*, Department.ManagerId  
FROM Emp INNER JOIN Department ON (Emp.DeptId = Department.DeptId);  
  
or  
  
SELECT \*  
FROM Emp NATURAL JOIN Department;

1. There is no loss of information: the definition of *lossless* decomposition.
2. No previously mentioned redundancy and anomaly.

**2. Methods for good database designs**

Two main tools:

1. Integrity Rules:  data constraint rules for avoiding data inconsistency.
2. Normal Forms:  a set of rules for designing good relation schemas.

**2.1 Integrity Rules**

* Most of the integrity rules are *application* dependent.
* Need to analyze the semantic of the applications to find out the integrity rules.
* These are known as *Database-Specific Integrity Rules*, or Application-Specific Integrity Rules.
  + Specific to the given application
  + Not universally applicable.

***Examples***: some database-specific integrity rules.

1. Student Id should be a seven-digit number.
2. Date of Birth should be greater than 1900.
3. The room number of Delta Building should start with a 'D'.
4. A student cannot take more than 24 credits in any semester.
5. A student must show proof of a meningitis shot before registration for the first semester.

**2.2 General Integrity Rules**

* They should be satisfied by *every* database.
* However, they are not necessarily enforced by the DBMS.
* Two general integrity rules in relational databases:
  1. *Entity Integrity Rule*: based on the concepts of primary keys (PK) and candidate keys (CK)
  2. *Referential Integrity Rule*: based on the concept of foreign keys (FK).

**2.3 Entity Integrity Rule**

* Entity Integrity: no*component* of a *candidate key* of a relation can have a null value.
* Meaning: In a relational database, a row that cannot be identified by ites CK will not be stored.

***Example:***

Employee(EmpId, Name, DeptId, Salary)

|  |  |  |  |
| --- | --- | --- | --- |
| **EmpId** | **Name** | **DeptId** | **Salary** |
| 101 | Lady Gaga | D123 | 55000000 |
| 122 | Brad Pitts | D123 | 10100000 |
| 140 | Lebron James | D123 | 50000000 |
| 155 | Narendra Modi | @: null | @ |
| *@* | Jennifer Lopez | D222 | 20000000***(this row cannot be added)*** |
| *@* | John Smiths | D300 | 70000***(this row cannot be added)*** |

* If EmpId is a candidate key, this Employee instance does not satisfy the entity integrity rule.
* Conversely, if we accept the relation instance above as valid, EmpId cannot be a candidate key.

**2.4 Referential Integrity Rule**

* Referential integrity rule: relations do not contain any *unmatched* *non-null foreign key* values.
* Any non-null value of a foreign key K must appear in the parent (referenced) relation where K is a candidate key.

***Example:***

EMP(EmpId, Name, DeptId)

|  |  |  |
| --- | --- | --- |
| **EmpId** | **Name** | **DeptId** |
| 101 | Lady Gaga | D123 |
| 122 | Brad Pitts | D123 |
| 140 | Lebron James | *@* |
| 155 | Narendra Modi | D222 |
| 167 | Jennifer Lopez | D222 |
| 311 | John Smiths | D300 |

Department(DeptId, ManagerId)

|  |  |
| --- | --- |
| **DeptId** | **ManagerId** |
| D123 | M10 |
| D222 | M21 |
| D300 | M33 |

* DeptId is a foreign key in the table EMP, referencing DeptId in the table Department. EMP(DeptId) references Department(DeptId).
* The referential integrity rule is satisfied.
* Note that DeptId may be null in EMP.

***Example:***

EMP(EmpId, Name, DeptId)

|  |  |  |
| --- | --- | --- |
| **EmpId** | **Name** | **DeptId** |
| 101 | Lady Gaga | D123 |
| 122 | Brad Pitts | D123 |
| 140 | Lebron James | D123 |
| 155 | Narendra Modi | D222 |
| 167 | Jennifer Lopez | D222 |
| 311 | John Smiths | D300 |
| **350** | **Bun Yue** | **D119*(should not be added)*** |

Department(DeptId, ManagerId)

|  |  |
| --- | --- |
| **DeptId** | **ManagerId** |
| D123 | M10 |
| D222 | M21 |
| D300 | M33 |

* The referential integrity rule is not satisfied in the example above.

Note:

1. In practical DBMS, pay attention to where the referential integrity rule is enforced.
2. For example, in MySQL, only the INNODB data engine may enforce the referential integrity rule.
3. If the DBMS does not enforce the referential integrity rule, it will be the task of the DB developers to do so.