Database Normalization

Prepared By: Dr. Mohammad Rezwanul Huq

Functional Dependencies

- A *Functional Dependency* describes a relationship between *attributes* within a single relation.
- An attribute is *functionally dependent* on another if we can use the value of one attribute to determine the value of another.
- Example: Employee_Name is functionally dependent on Social_Security_Number because Social_Security_Number can be used to uniquely determine the value of Employee_Name.
- We use the arrow symbol → to indicate a functional dependency.

  X → Y is read *X functionally determines Y*

  Here are a few more examples:

```
Student_ID → Student_Major
Student_ID, CourseNumber, Semester → Grade
Course_Number, Section → Professor, Classroom, NumberOfStudents
SKU → Compact_Disk_Title, Artist
CarModel, Options, TaxRate → Car_Price
```

Keys and Uniqueness

- **Key**: One or more attributes that uniquely identify a tuple (row) in a relation.
- The selection of keys will depend on the particular application being considered.
- In most cases the key for a relation will already be specified during the conversion from the E-R model to a set of relations.
- Users can also offer some guidance as to what would make an appropriate key.
- Recall that no two relations should have exactly the same values, thus a candidate key would consist of all of the attributes in a relation.
- A key functionally determines a tuple (row). So one functional dependency that can always be written is:

```
The Key → All other attributes
```

Modification Anomalies

- Once our E-R model has been converted into relations, we may find that some relations are not properly specified. There can be a number of problems:

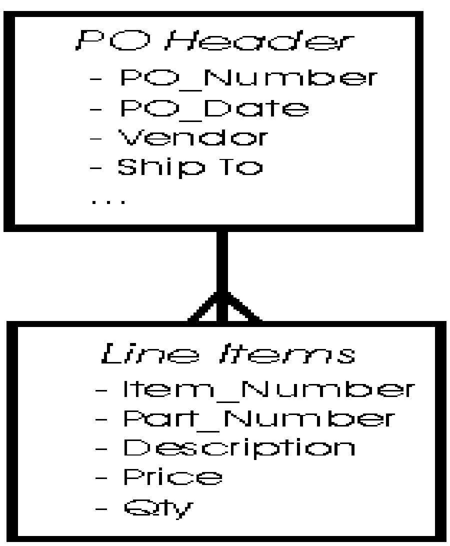
- **Deletion Anomaly**: Deleting one fact or data point from a relation results in other information being lost.
- **Insertion Anomaly**: Inserting a new fact or tuple into a relation requires we have information from two or more entities – this situation might not be feasible.
- **Update Anomaly**: Updating one fact in a relation requires us to update multiple tuples.

- Here is a quick example to illustrate these anomalies: A company has a Purchase Order form:


Purchase Order
PO#: 0101 Date: 9/24/97
Vendor: Happy V Co.
Ship to: 123 Cool Way

Ln	Part	Desc	Price	Qty
1	P99	Plate	$3.00	7
2	P98	Cup	$1.00	11

- Our dutiful consultant creates the E-R Model directly matching the purchase order:



When we follow the steps to convert to a set of relations this results in two relations (keys are underlined):

```
PO_HEADER (PO_Number, PODate, Vendor, Ship_To, ...)

LINE_ITEMS (PO_Number, ItemNum, PartNum, Description, Price, Qty)
```

- Consider some sample data for the LINE_ITEMS relation:

PO_Number	**ItemNum**	**PartNum**	**Description**	**Price**	**Qty**
O101	I01	P99	Plate	$3.00	7
O101	I02	P98	Cup	$1.00	11
O101	I03	P77	Bowl	$2.00	6
O102	I01	P99	Plate	$3.00	5
O102	I02	P77	Bowl	$2.00	5
O103	I01	P33	Fork	$2.50	8

- What are some of the problems with this relation ?

1. What happens if we want to add the fact that Order O103 has quantity 5 of part P99 ?
2. What happens when we delete item I02 from Order O101 ?
3. What happens if we want to change the price of the Plate (P99)?

- These problems occur because the relation in question contains data about 2 or more *themes*.
- Typical way to solve these anomalies is to split the relation in to two or more relations – This is part of the *Process* called *Normalization* discussed next.

Normalization Process

- Relations can fall into one or more categories (or classes) called *Normal Forms*
- **Normal Form**: A class of relations free from a certain set of modification anomalies.
- Normal forms are given names such as:
  - First normal form (1NF)
  - Second normal form (2NF)
  - Third normal form (3NF)
  - Boyce-Codd normal form (BCNF)
  - Fourth normal form (4NF)
  - Fifth normal form (5NF)
  - Domain-Key normal form (DK/NF)
- These forms are cumulative. A relation in Third normal form is also in 2NF and 1NF.
- The *Normalization Process* for a given relation consists of:

- Specify the *Key* of the relation
- Specify the *functional dependencies* of the relation.
  Sample data (tuples) for the relation can assist with this step.
- Apply the definition of each normal form (starting with 1NF).
- If a relation fails to meet the definition of a normal form, change the relation (most often by splitting the relation into two new relations) until it meets the definition.
- Re-test the modified/new relations to ensure they meet the definitions of each normal form.

In the next set of notes, each of the normal forms will be defined along with an example of the normalization steps.

First Normal Form (1NF)

- A relation is in first normal form if it meets the definition of a relation:
  1. Each attribute (column) value must be a single value only.
  2. All values for a given attribute (column) must be of the same type.
  3. Each attribute (column) name must be unique.
  4. The order of attributes (columns) is insignificant
  5. No two tuples (rows) in a relation can be identical.
  6. The order of the tuples (rows) is insignificant.
- If you have a *key* defined for the relation, then you can meet the *unique row* requirement.
- Example relation in 1NF (note that key attributes are underlined):

  STOCKS (Company, <u>Symbol</u>, Headquarters, <u>Date</u>, Close_Price)

Company	<u>Symbol</u>	Headquarters	<u>Date</u>	Close Price
Microsoft	MSFT	Redmond, WA	09/07/2013	23.96
Microsoft	MSFT	Redmond, WA	09/08/2013	23.93
Microsoft	MSFT	Redmond, WA	09/09/2013	24.01
Oracle	ORCL	Redwood Shores, CA	09/07/2013	24.27
Oracle	ORCL	Redwood Shores, CA	09/08/2013	24.14
Oracle	ORCL	Redwood Shores, CA	09/09/2013	24.33

Note that the key (which consists of the <u>Symbol</u> and the <u>Date</u>) can uniquely determine the Company, headquarters and Close Price of the stock. Here was assume that Symbol must be unique but Company, Headquarters, Date and Price are not unique

Second Normal Form (2NF)

- A relation is in second normal form (2NF) if all of its non-key attributes are dependent on all of the *key*.
- Another way to say this: A relation is in second normal form if it is free from partial-key dependencies
- Relations that have a single attribute for a key are automatically in 2NF.
- This is one reason why we often use artificial identifiers (non-composite keys) as keys.
- In the example below, Close Price is dependent on Company, Date
- The following example relation *is not* in 2NF:

  STOCKS (Company, <u>Symbol</u>, Headquarters, <u>Date</u>, Close_Price)

Company	**<u>Symbol</u>**	**Headquarters**	**<u>Date</u>**	**Close Price**
Microsoft	MSFT	Redmond, WA	09/07/2013	23.96
Microsoft	MSFT	Redmond, WA	09/08/2013	23.93
Microsoft	MSFT	Redmond, WA	09/09/2013	24.01
Oracle	ORCL	Redwood Shores, CA	09/07/2013	24.27
Oracle	ORCL	Redwood Shores, CA	09/08/2013	24.14
Oracle	ORCL	Redwood Shores, CA	09/09/2013	24.33

- To start the normalization process, list the functional dependencies (FD):

```
FD1: Symbol, Date → Company, Headquarters, Close Price
FD2: Symbol → Company, Headquarters
```

- Consider that Symbol, Date → Close Price.

  So we might use <u>Symbol, Date</u> as our key.

- However we also see that: Symbol → Headquarters
- This violates the rule for 2NF in that a *part of our key* key determines a non-key attribute.
- Another name for this is a *Partial key dependency*. Symbol is only a "part" of the key and it determines a non-key attribute.
- Also, consider the insertion and deletion anomalies.
- **One Solution:** Split this up into two new relations:

  COMPANY (Company, <u>Symbol</u>, Headquarters)

  STOCK_PRICES (<u>Symbol</u>, <u>Date</u>, Close_Price)

- At this point we have two new relations in our relational model. The original “STOCKS” relation we started with is removed form the model.
- Sample data and functional dependencies for the two new relations:
- COMPANY Relation:

Company	**Symbol**	**Headquarters**
Microsoft	MSFT	Redmond, WA
Oracle	ORCL	Redwood Shores, CA

- `FD1: Symbol → Company, Headquarters`

- STOCK_PRICES relation:

Symbol	**Date**	**Close Price**
MSFT	09/07/2013	23.96
MSFT	09/08/2013	23.93
MSFT	09/09/2013	24.01
ORCL	09/07/2013	24.27
ORCL	09/08/2013	24.14
ORCL	09/09/2013	24.33

`FD1: Symbol, Date → Close Price`

- In checking these new relations we can confirm that they meet the definition of 1NF (each one has well defined unique keys) and 2NF (no partial key dependencies).

Third Normal Form (3NF)

- A relation is in third normal form (3NF) if it is in second normal form and it contains no *transitive dependencies*.
- Consider relation R containing attributes A, B and C. R(A, B, C)
- If A → B and B → C then A → C
- **Transitive Dependency**: Three attributes with the above dependencies.
- Example: At CUNY:

```
Course_Code → Course_Number, Section
Course_Number, Section → Classroom, Professor
```

- Consider one of the new relations we created in the STOCKS example for 2nd normal form:

Company	**Symbol**	**Headquarters**
Microsoft	MSFT	Redmond, WA
Oracle	ORCL	Redwood Shores, CA

- The functional dependencies we can see are:

```
FD1: Symbol  →   Company
FD2: Company → Headquarters
so therefore:
Symbol → Headquarters
```

- This is a transitive dependency.
- What happens if we remove Oracle?

  We loose information about 2 different facts.

- The solution again is to split this relation up into two new relations:

  STOCK_SYMBOLS(Company, Symbol)

  COMPANY_HEADQUARTERS(Company, Headquarters)

- This gives us the following sample data and FD for the new relations

Company	**Symbol**
Microsoft	MSFT
Oracle	ORCL

```
FD1: Symbol → Company
```

	Company	**Headquarters**
	Microsoft	Redmond, WA
	Oracle	Redwood Shores, CA

```
FD1:  Company →  Headquarters
```

- Again, each of these new relations should be checked to ensure they meet the definition of 1NF, 2NF and now 3NF.

Boyce-Codd Normal Form (BCNF)

- A relation is in BCNF if every determinant is a candidate key.

- Recall that not all determinants are keys.
- Those determinants that are keys we initially call *candidate keys*.
- Eventually, we select a single candidate key to be *the key* for the relation.
- Consider the following example:
    - Funds consist of one or more Investment Types.
    - Funds are managed by one or more Managers
    - Investment Types can have one more Managers
    - Managers only manage one type of investment.
- Relation: FUNDS (FundID, InvestmentType, Manager)

FundID	**InvestmentType**	**Manager**
99	Common Stock	Smith
99	Municipal Bonds	Jones
33	Common Stock	Green
22	Growth Stocks	Brown
11	Common Stock	Smith

```
FD1:  FundID, InvestmentType → Manager
FD2:  FundID, Manager        → InvestmentType
FD3:  Manager                → InvestmentType
```

- In this case, the combination FundID and InvestmentType form a *candidate key* because we can use FundID,InvestmentType to uniquely identify a tuple in the relation.
- Similarly, the combination FundID and Manager also form a *candidate key* because we can use FundID, Manager to uniquely identify a tuple.
- Manager by itself is not a candidate key because we cannot use Manager alone to uniquely identify a tuple in the relation.
- Is this relation FUNDS(FundID, InvestmentType, Manager) in 1NF, 2NF or 3NF ?

  Given we pick FundID, InvestmentType as the *Primary Key:* 1NF for sure.

  2NF because all of the non-key attributes (Manager) is dependant on all of the key.

  3NF because there are no transitive dependencies.

- However consider what happens if we delete the tuple with FundID 22. We loose the fact that Brown manages the InvestmentType "Growth Stocks."
- Therefore, while FUNDS relation is in 1NF, 2NF and 3NF, it is in BCNF because not all determinants (Manager in FD3) are candidate keys.
- The following are steps to normalize a relation into BCNF:
    - List all of the determinants.

    - See if each determinant can act as a key (candidate keys).
    - For any determinant that is *not* a candidate key, create a new relation from the functional dependency. Retain the determinant in the original relation.
- For our example:

  FUNDS (FundID, InvestmentType, Manager)

The determinants are:

```
FundID, InvestmentType
FundID, Manager
Manager
```

1. Which determinants can act as keys ?

```
FundID, InvestmentType YES
FundID, Manager YES
Manager NO
```

2. Create a new relation from the functional dependency:
   MANAGERS(Manager, InvestmentType)

   FUND_MANAGERS(FundID, Manager)

   In this last step, we have retained the determinant "Manager" in the original relation MANAGERS.

- Each of the new relations sould be checked to ensure they meet the definitions of 1NF, 2NF, 3NF and BCNF

Fourth Normal Form (4NF)

- A relation is in fourth normal form if it is in BCNF and it contains no *multivalued dependencies*.
- **Multivalued Dependency**: A type of functional dependency where the determinant can determine more than one value.
- More formally, there are 3 criteria:
    1. There must be at least 3 attributes in the relation. call them A, B, and C, for example.
    2. Given A, one can determine multiple values of B.

       Given A, one can determine multiple values of C.

    3. B and C are independent of one another.
- Book example:

  Student has one or more majors.

  Student participates in one or more activities.

StudentID	Major	Activities
100	CIS	Baseball
100	CIS	Volleyball
100	Accounting	Baseball
100	Accounting	Volleyball
200	Marketing	Swimming

- `FD1: StudentID →→ Major`

```
FD2: StudentID →→ Activities
```

Portfolio ID	Stock Fund	Bond Fund
999	Janus Fund	Municipal Bonds
999	Janus Fund	Dreyfus Short-Intermediate Municipal Bond Fund
999	Scudder Global Fund	Municipal Bonds
999	Scudder Global Fund	Dreyfus Short-Intermediate Municipal Bond Fund
888	Kaufmann Fund	T. Rowe Price Emerging Markets Bond Fund

- A few characteristics:

1. No regular functional dependencies
2. All three attributes taken together form the key.
3. Latter two attributes are independent of one another.
4. Insertion anomaly: Cannot add a stock fund without adding a bond fund (NULL Value). Must always maintain the combinations to preserve the meaning.

- Stock Fund and Bond Fund form a multivalued dependency on Portfolio ID.

```
PortfolioID   →→   Stock Fund
PortfolioID   →→   Bond Fund
```

- Resolution: Split into two tables with the common key:

Portfolio ID	Stock Fund
999	Janus Fund
999	Scudder Global Fund
888	Kaufmann Fund

Portfolio ID	Bond Fund
999	Municipal Bonds
999	Dreyfus Short-Intermediate Municipal Bond Fund
888	T. Rowe Price Emerging Markets Bond Fund

All-in-One Database Normalization Example

Many of you asked for a “complete” example that would run through all of the normal forms from beginning to end using the same tables. This is tough to do, but here is an attempt:

Example relation:

EMPLOYEE (<u>Name</u>, <u>Project</u>, <u>Task</u>, Office, Floor, Phone)

Note: Keys are underlined.

Example Data:

<u>Name</u>	**<u>Project</u>**	**<u>Task</u>**	**Office**	**Floor**	**Phone**
Bill	100X	T1	400	4	1400
Bill	100X	T2	400	4	1400
Bill	200Y	T1	400	4	1400
Bill	200Y	T2	400	4	1400
Sue	100X	T33	442	4	1442
Sue	200Y	T33	442	4	1442
Sue	300Z	T33	442	4	1442
Ed	100X	T2	588	5	1588

- **Name** is the employee’s name
- **Project** is the project they are working on. Bill is working on two different projects, Sue is working on 3.
- **Task** is the current task being worked on. Bill is now working on Tasks T1 and T2. Note that Tasks are independent of the project. Examples of a task might be faxing a memo or holding a meeting.
- **Office** is the office number for the employee. Bill works in office number 400.
- **Floor** is the floor on which the office is located.
- **Phone** is the phone extension. Note this is associated with the phone in the given office.

First Normal Form

- Assume the **key** is <u>Name</u>, <u>Project</u>, <u>Task</u>.
- Is EMPLOYEE in 1NF ?

Second Normal Form

- List all of the functional dependencies for EMPLOYEE.

- Are all of the non-key attributes dependant on *all* of the key ?
- It seems if we know the employee's name, we can figure out their office, floor and phone.
- Split into two relations EMPLOYEE_PROJECT_TASK and EMPLOYEE_OFFICE_PHONE.

`EMPLOYEE_PROJECT_TASK (Name, Project, Task)`

Name	Project	Task
Bill	100X	T1
Bill	100X	T2
Bill	200Y	T1
Bill	200Y	T2
Sue	100X	T33
Sue	200Y	T33
Sue	300Z	T33
Ed	100X	T2

`EMPLOYEE_OFFICE_PHONE (Name, Office, Floor, Phone)`

Name	Office	Floor	Phone
Bill	400	4	1400
Sue	442	4	1442
Ed	588	5	1588

Third Normal Form

- Assume each office has exactly one phone number.
- Are there any transitive dependencies ?
- Where are the modification anomalies in EMPLOYEE_OFFICE_PHONE ?
- Split EMPLOYEE_OFFICE_PHONE into two new relations.

`EMPLOYEE_PROJECT_TASK (Name, Project, Task)`

Name	Project	Task
Bill	100X	T1
Bill	100X	T2
Bill	200Y	T1
Bill	200Y	T2
Sue	100X	T33
Sue	200Y	T33

Sue	300Z	T33
Ed	100X	T2

EMPLOYEE_OFFICE (Name, Office, Floor)

Name	**Office**	**Floor**
Bill	400	4
Sue	442	4
Ed	588	5

EMPLOYEE_PHONE (Office, Phone)

Office	**Phone**
400	1400
442	1442
588	1588

Boyce-Codd Normal Form

- List all of the functional dependencies for EMPLOYEE_PROJECT_TASK, EMPLOYEE_OFFICE and EMPLOYEE_PHONE. Look at the determinants.
- Are all determinants candidate keys ?

Forth Normal Form

- Are there any multivalued dependencies ?
- What are the modification anomalies ?
- Split EMPLOYEE_PROJECT_TASK.

EMPLOYEE_PROJECT (Name, Project)

Name	**Project**
Bill	100X
Bill	200Y
Sue	100X
Sue	200Y
Sue	300Z
Ed	100X

EMPLOYEE_TASK (Name, Task)

Name	Task
Bill	T1
Bill	T2
Sue	T33
Ed	T2

```
EMPLOYEE_OFFICE (Name, Office, Floor)
```

Name	Office	Floor
Bill	400	4
Sue	442	4
Ed	588	5

OFFICE_PHONE (Office, Phone)

Office	Phone
400	1400
442	1442
588	1588

At each step of the process, we did the following:

1. Write out the relation
2. (optionally) Write out some example data.
3. Write out all of the functional dependencies
4. Starting with 1NF, go through each normal form and state why the relation is in the given normal form.

Another short example

Consider the following example of normalization for a CUSTOMER relation.

Relation Name

CUSTOMER (CustomerID, Name, Street, City, State, Zip, Phone)

Example Data

CustomerID	Name	Street	City	State	Zip	Phone
C101	Bill Smith	123 First St.	New Brunswick	NJ	07101	732-555-1212
C102	Mary Green	11 Birch St.	Old Bridge	NJ	07066	908-555-1212

Functional Dependencies

```
FD1: CustomerID → Name, Street, City, State, Zip, Phone
FD2: Zip → City, State
```

Normalization

- **1NF** Meets the definition of a relation.
- **2NF** All non key attributes are dependent on all of the key.
- **3NF** Relation CUSTOMER is not in 3NF because there is a transitive dependency.
  CustomerID → Zip and Zip → City, State

  Solution: Split CUSTOMER into two relations:

  `CUSTOMER (`<u>`CustomerID`</u>`, Name, Street, Zip, Phone)`

  `ZIPCODES (`<u>`Zip`</u>`, City, State)`

  Check both CUSTOMER and ZIPCODE to ensure they are both in 1NF up to BCNF.