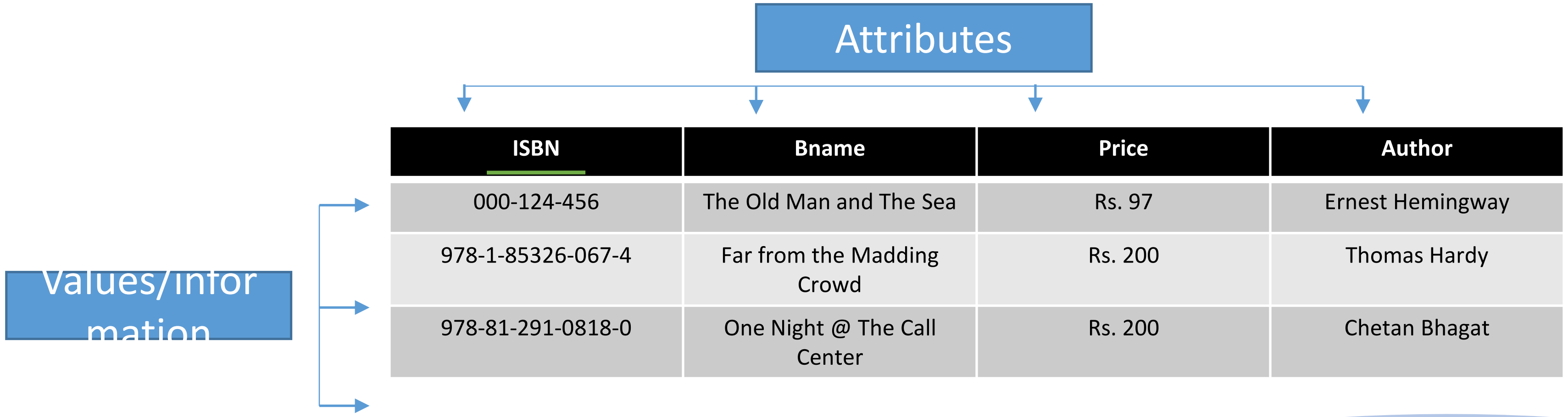
Chapter 4: Data Constraints and Normalization

Relational Database Model

- Relational Model was developed by E.F. Codd,
- Most popular database model in the world,
- The data are stored on relation (table), relation in Relational Database is similar to table where column represents attributes, and information are stored in rows.



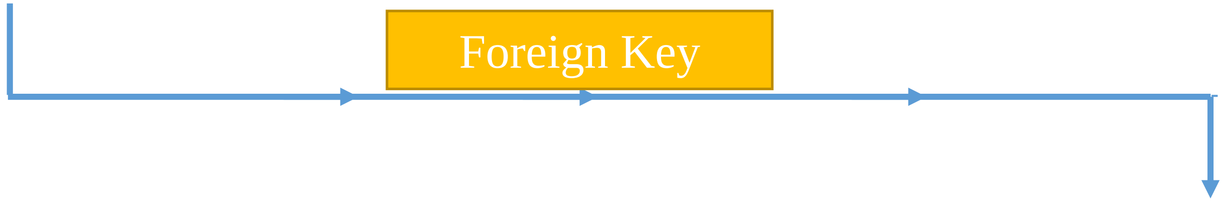
ISBN	Bname	Price	Author
000-124-456	The Old Man and The Sea	Rs. 97	Ernest Hemingway
978-1-85326-067-4	Far from the Madding Crowd	Rs. 200	Thomas Hardy
978-81-291-0818-0	One Night @ The Call Center	Rs. 200	Chetan Bhagat

Contd…

- The concept of Primary Key and Foreign Key helps to create logical relationship between relations.
- Primary Key is a one of the best keys that is chosen by database designer for the purpose of uniquely identifying all the entities of entity set.
  - A combination of a NOT NULL and UNIQUE. Ensures that a column (or combination of two or more columns) have an unique identity which helps to find a particular record in a table more easily and quickly.
- Foreign Key is actually the primary key of one table and serves as attribute for another table.
  - Ensures the referential integrity of the data in one table to match values in another table

Contd…

ISBN	Bname	Price	Author
000-124-456	The Old Man and The Sea	Rs. 97	Ernest Hemingway
978-1-85326-067-4	Far from the Madding Crowd	Rs. 200	Thomas Hardy
978-81-291-0818-0	One Night @ The Call Center	Rs. 200	Chetan Bhagat



ID	Name	Grade	ISBN
0001	Sanjay Sharma	BBA	978-81-291-0818-0
0002	Sushil Shrestha	BSC	000-124-456
0030	Samikshaya Sharma	BBA	978-1-85326-067-4

Data Types and their Notation in SQL SERVER

- Integer : INT
- Number (Decimal) : FLOAT or REAL
- Currency : MONEY
- String (fixed) : CHAR
- String (Variable) : VARCHAR
- Date : DATETIME

Data integrity

- The fundamental set of rules,
- Implemented for the purpose of maintaining accuracy, reliability, and consistency in data,
- Helps to avoid accidental deletion of data,
- Prevents the entry of invalid data in database,

Types of Data Integrity

Entity Integrity

- Generally applies on an attribute,
- Concerned with the presence of primary key in each relation(table),
- It advocates the following:
    - Primary Key must be NOT NULL,
    - Primary Key must be UNIQUE,

NOTE:

IF THERE IS NULL VALUE FOR A PRIMARY KEY, THEN IT WILL BE UNATTAINABLE FOR US TO IDENTIFY ALL THE TUPLES INDIVIDUALLY.

IN SQL (Entity Integrity)

```
CREATE TABLE Library (
ISBN INT,
Bname VARCHAR (20),
Price MONEY,
Author VARCHAR (20),
CONSTRAINT pk_id PRIMARY KEY (ISBN));
```

Primary Key

ISBN	Bname	Price	Author
000-124-456	The Old Man and The Sea	Rs. 97	Ernest Hemingway
978-1-85326-067-4	Far from the Madding Crowd	Rs. 200	Thomas Hardy
978-81-291-0818-0	One Night @ The Call Center	Rs. 200	Chetan Bhagat

Contd…

- Domain Integrity
  - Domain Constraint is one of the elementary form of integrity constraint that helps to maintain accuracy and consistency,
  - Helps to avoid duplication of data,
- Referential Integrity
  - Enables to establish relationship between two relations through the application of concept of PRIMARY KEY and FOREIGN KEY.

Introduction to Integrity Constraints

Integrity Constraint

- Set of rules that helps to maintain correctness of data,
- Prevent from accidental deletion and insertion of data,
- Types of Integrity constraint
    - Domain Constraint
    - Referential Integrity Constraint
    - Assertion
    - Triggers

Domain Constraints

- Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency.
- Domain constraints are the most elementary form of integrity constraint.
- They test values inserted in the database, and test queries to ensure that the comparisons make sense.
- New domains can be created from existing data types
  - E.g. **create domain** *Dollars* **numeric**(12, 2)
    create domain *Pounds* **numeric**(12,2)
- We cannot assign or compare a value of type Dollars to a value of type Pounds.
  - However, we can convert type as below
    (**cast** *r.A* **as** *Pounds*)
    (Should also multiply by the dollar-to-pound conversion-rate)

Domain Constraints (Cont.)

- The **check** clause in SQL permits domains to be restricted:
  - Use **check** clause to ensure that an hourly-wage domain allows only values greater than a specified value.

    create domain *hourly-wage* **numeric(5,2)**

    constraint *value-test* **check**(*value* > = 4.00)
  - The domain has a constraint that ensures that the hourly-wage is greater than 4.00
  - The clause **constraint** *value-test* is optional; useful to indicate which constraint an update violated.
- Can have complex conditions in domain check
  - **create domain** *AccountType* **char**(10)
    constraint *account-type-test*
    check (**value in** ('Checking', 'Saving'))
  - **check** (*branch-name* **in** (**select** *branch-name* **from** *branch*))

Referential Integrity

- Ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation.
  - Example: If "Perryridge" is a branch name appearing in one of the tuples in the *account* relation, then there exists a tuple in the *branch* relation for branch "Perryridge".
- Formal Definition
  - Let $r_1(R_1)$ and $r_2(R_2)$ be relations with primary keys K_1 and K_2 respectively.
  - The subset α of R_2 is a ***foreign key*** referencing K_1 in relation r_1, if for every t_2 in r_2 there must be a tuple t_1 in r_1 such that $t_1[K_1] = t_2[\alpha]$.
  - Referential integrity constraint also called subset dependency since its can be written as

$$\Pi_{\alpha}(r_2) \subseteq \Pi_{K1}(r_1)$$

Referential Integrity in SQL

- Primary and candidate keys and foreign keys can be specified as part of the SQL **create table** statement:
  - The **primary key** clause lists attributes that comprise the primary key.
  - The **unique key** clause lists attributes that comprise a candidate key.
  - The **foreign key** clause lists the attributes that comprise the foreign key and the name of the relation referenced by the foreign key.
- By default, a foreign key references the primary key attributes of the referenced table

  foreign key (*account-number*) **references** *account*
- Short form for specifying a single column as foreign key

  account-number **char** (10) **references** *account*
- Reference columns in the referenced table can be explicitly specified
  - but must be declared as primary/candidate keys

  foreign key (*account-number*) **references** *account*(*account-number*)

Referential Integrity in SQL – Example

```
create table customer
  (customer-name  char(20),
  customer-street  char(30),
  customer-city  char(30),
  primary key (customer-name))
create table branch
  (branch-name  char(15),
  branch-city      char(30),
  assets              integer,
  primary key (branch-name))
```

Referential Integrity in SQL – Example (Cont.)

```
create table account
   (account-number   char(10),
   branch-name   char(15),
   balance           integer,
   primary key (account-number),
   foreign key (branch-name) references branch)

create table depositor
   (customer-name    char(20),
   account-number    char(10),
   primary key (customer-name, account-number),
   foreign key (account-number) references account,
   foreign key (customer-name) references customer)
```

Cascading Actions in SQL

```
create table account
    . . .
    foreign key(branch-name)    references branch
                                on delete cascade
                                on update cascade
    . . . )
```

- Due to the **on delete cascade** clauses, if a delete of a tuple in *branch* results in referential-integrity constraint violation, the delete “cascades” to the *account* relation, deleting the tuple that refers to the branch that was deleted.
- Cascading updates are similar.

Cascading Actions in SQL (Cont.)

- If there is a chain of foreign-key dependencies across multiple relations, with **on delete cascade** specified for each dependency, a deletion or update at one end of the chain can propagate across the entire chain.
- If a cascading update to delete causes a constraint violation that cannot be handled by a further cascading operation, the system aborts the transaction.
  - As a result, all the changes caused by the transaction and its cascading actions are undone.
- Referential integrity is only checked at the end of a transaction
  - Intermediate steps are allowed to violate referential integrity provided later steps remove the violation
  - Otherwise it would be impossible to create some database states, e.g. insert two tuples whose foreign keys point to each other
    - E.g. *spouse* attribute of relation
      marriedperson(name, address, spouse)

Referential Integrity in SQL (Cont.)

- Alternative to cascading:
  - **on delete set null**
  - **on delete set default**
- Null values in foreign key attributes complicate SQL referential integrity semantics, and are best prevented using **not null**
  - if any attribute of a foreign key is null, the tuple is defined to satisfy the foreign key constraint!

Assertions

- An *assertion* is a predicate expressing a condition that we wish the database always to satisfy.
- An assertion in SQL takes the form

  create assertion <assertion-name> **check** <predicate>
- When an assertion is made, the system tests it for validity, and tests it again on every update that may violate the assertion
  - This testing may introduce a significant amount of overhead; hence assertions should be used with great care.
- Asserting

  for all X, P(X)

  is achieved in a round-about fashion using

  not exists X such that not P(X)

Assertion Example

- The sum of all loan amounts for each branch must be less than the sum of all account balances at the branch.

```
create assertion sum-constraint check
  (not exists (select * from branch
               where (select sum(amount) from loan
                      where loan.branch-name =
                            branch.branch-name)
                  >= (select sum(amount) from account
                      where loan.branch-name =
                            branch.branch-name)))
```

Assertion Example

- Every loan has at least one borrower who maintains an account with a minimum balance or $1000.00

```
create assertion balance-constraint check
   (not exists (
        select * from loan
         where not exists (
            select *
             from borrower, depositor, account
             where loan.loan-number = borrower.loan-number
                 and borrower.customer-name = depositor.customer-name
                 and depositor.account-number = account.account-number
                 and account.balance >= 1000)))
```

Triggers

- A **trigger** is a statement that is executed automatically by the system as a side effect of a modification to the database.
- To design a trigger mechanism, we must:
  - Specify the conditions under which the trigger is to be executed.
  - Specify the actions to be taken when the trigger executes.
- Triggers introduced to SQL standard in SQL, but supported even earlier using non-standard syntax by most databases.

Trigger Example

- Suppose that instead of allowing negative account balances, the bank deals with overdrafts by
  - setting the account balance to zero
  - creating a loan in the amount of the overdraft
  - giving this loan a loan number identical to the account number of the overdrawn account
- The condition for executing the trigger is an update to the *account* relation that results in a negative *balance* value.

Trigger Example in SQL

```
create trigger overdraft-trigger after update on account
referencing new row as nrow  for each row
when nrow.balance < 0
begin atomic
       insert into borrower
         (select customer-name, account-number
          from depositor
          where nrow.account-number =
                   depositor.account-number);
      insert into loan values
         (n.row.account-number, nrow.branch-name,
                                              – nrow.balance);
      update account set balance = 0
       where account.account-number = nrow.account-number
end
```

Triggering Events and Actions in SQL

- Triggering event can be **insert**, **delete** or **update**
- Triggers on update can be restricted to specific attributes
  - **E.g. create trigger** *overdraft-trigger* **after update of** *balance* **on** *account*
- Values of attributes before and after an update can be referenced
  - **referencing old row as** : for deletes and updates
  - **referencing new row as :** for inserts and updates
- Triggers can be activated before an event, which can serve as extra constraints. E.g. convert blanks to null.

```
create trigger setnull-trigger before update on r
referencing new row as nrow
for each row
   when nrow.phone-number = ' '
   set nrow.phone-number = null
```

Statement Level Triggers

- Instead of executing a separate action for each affected row, a single action can be executed for all rows affected by a transaction
  - Use **for each statement** instead of **for each row**
  - Use **referencing old table** or **referencing new table** to refer to temporary tables (called ***transition tables***) containing the affected rows
  - Can be more efficient when dealing with SQL statements that update a large number of rows

When Not To Use Triggers

- Triggers were used earlier for tasks such as
  - maintaining summary data (e.g. total salary of each department)
  - Replicating databases by recording changes to special relations (called **change** or **delta** relations) and having a separate process that applies the changes over to a replica
- There are better ways of doing these now:
  - Databases today provide built in materialized view facilities to maintain summary data
  - Databases provide built-in support for replication
- Encapsulation facilities can be used instead of triggers in many cases
  - Define methods to update fields
  - Carry out actions as part of the update methods instead of through a trigger

Security

- **Security** - protection from malicious attempts to steal or modify data.
  - Database system level
    - Authentication and authorization mechanisms to allow specific users access only to required data
    - We concentrate on authorization in the rest of this chapter
  - Operating system level
    - Operating system super-users can do anything they want to the database! Good operating system

      level security is required.
  - Network level: must use encryption to prevent
    - Eavesdropping (unauthorized reading of messages)
    - Masquerading (pretending to be an authorized user or sending messages supposedly from authorized users)

Security (Cont.)

- Physical level
  - Physical access to computers allows destruction of data by intruders; traditional lock-and-key security is needed
  - Computers must also be protected from floods, fire, etc.
- Human level
  - Users must be screened to ensure that an authorized users do not give access to intruders
  - Users should be trained on password selection and secrecy

Authorization

Forms of authorization on parts of the database:

- **Read authorization** - allows reading, but not modification of data.
- **Insert authorization** - allows insertion of new data, but not modification of existing data.
- **Update authorization** - allows modification, but not deletion of data.
- **Delete authorization** - allows deletion of data

Authorization (Cont.)

Forms of authorization to modify the database schema:

- **Index authorization** - allows creation and deletion of indices.
- **Resources authorization** - allows creation of new relations.
- **Alteration authorization** - allows addition or deletion of attributes in a relation.
- **Drop authorization** - allows deletion of relations.

Granting of Privileges

- The passage of authorization from one user to another may be represented by an authorization graph.
- The nodes of this graph are the users.
- The root of the graph is the database administrator.
- Consider graph for update authorization on loan.
- An edge $U_i \rightarrow U_j$ indicates that user U_i has granted update authorization on loan to $U_{j.}$

$$\begin{tikzcd}
 & U_1 \arrow[r] \arrow[rrd] & U_4 & \\
DBA \arrow[ru] \arrow[r] \arrow[rd] & U_2 \arrow[rr] & & U_5 \\
 & U_3 & &
\end{tikzcd}$$

Authorization Grant Graph

- *Requirement*: All edges in an authorization graph must be part of some path originating with the database administrator
- If DBA revokes grant from U_1:
  - Grant must be revoked from U_4 since U_1 no longer has authorization
  - Grant must not be revoked from U_5 since U_5 has another authorization path from DBA through U_2
- Must prevent cycles of grants with no path from the root:
  - DBA grants authorization to U_7
  - U7 grants authorization to U_8
  - U8 grants authorization to U_7
  - DBA revokes authorization from U_7
- Must revoke grant U_7 to U_8 and from U_8 to U_7 since there is no path from DBA to U_7 or to U_8 anymore.

Security Specification in SQL

- The grant statement is used to confer authorization

  grant <privilege list>

  on <relation name or view name> to <user list>

- <user list> is:
  - a user-id
  - *public*, which allows all valid users the privilege granted
  - A role (more on this later)
- Granting a privilege on a view does not imply granting any privileges on the underlying relations.
- The grantor of the privilege must already hold the privilege on the specified item (or be the database administrator).

Privileges in SQL

- **select:** allows read access to relation,or the ability to query using the view
  - Example: grant users U_1, U_2, and U_3 **select** authorization on the *branch* relation:

    grant select on *branch* **to** U_1, U_2, U_3
- **insert**: the ability to insert tuples
- **update**: the ability to update using the SQL update statement
- **delete**: the ability to delete tuples.
- **references**: ability to declare foreign keys when creating relations.
- **usage**: In SQL-92; authorizes a user to use a specified domain
- **all privileges**: used as a short form for all the allowable privileges

Privilege To Grant Privileges

- **with grant option**: allows a user who is granted a privilege to pass the privilege on to other users.
  - Example:

    grant select on *branch* **to** U_1 **with grant option**

    gives U_1 the **select** privileges on branch and allows U_1 to grant this privilege to others

Roles

- Roles permit common privileges for a class of users can be specified just once by creating a corresponding “role”
- Privileges can be granted to or revoked from roles, just like user
- Roles can be assigned to users, and even to other roles
- SQL:1999 supports roles

```
create role teller
create role manager

grant select on branch to teller
grant update (balance) on account to teller
grant all privileges on account to manager

grant teller to manager

grant teller to alice, bob
grant manager to avi
```

Revoking Authorization in SQL

- The **revoke** statement is used to revoke authorization.

  revoke<privilege list>

  on <relation name or view name> **from** <user list> [**restrict**|**cascade**]

- Example:

  revoke select on *branch* **from** U_1, U_2, U_3 **cascade**

- Revocation of a privilege from a user may cause other users also to lose that privilege; referred to as cascading of the **revoke**.
- We can prevent cascading by specifying **restrict**:

  revoke select on *branch* **from** U_1, U_2, U_3 **restrict**

  With **restrict**, the **revoke** command fails if cascading revokes are required.

Revoking Authorization in SQL (Cont.)

- <privilege-list> may be **all to** revoke all privileges the revokee may hold.
- If <revokee-list> includes **public** all users lose the privilege except those granted it explicitly.
- If the same privilege was granted twice to the same user by different grantees, the user may retain the privilege after the revocation.
- All privileges that depend on the privilege being revoked are also revoked.

Limitations of SQL Authorization

- SQL does not support authorization at a tuple level
  - E.g. we cannot restrict students to see only (the tuples storing) their own grades
- With the growth in Web access to databases, database accesses come primarily from application servers.
  - End users don't have database user ids, they are all mapped to the same database user id
- All end-users of an application (such as a web application) may be mapped to a single database user
- The task of authorization in above cases falls on the application program, with no support from SQL
  - Benefit: fine grained authorizations, such as to individual tuples, can be implemented by the application.
  - Drawback: Authorization must be done in application code, and may be dispersed all over an application
  - Checking for absence of authorization loopholes becomes very difficult since it requires reading large amounts of application code

Audit Trails

- An audit trail is a log of all changes (inserts/deletes/updates) to the database along with information such as which user performed the change, and when the change was performed.
- Used to track erroneous/fraudulent updates.
- Can be implemented using triggers, but many database systems provide direct support.

Authentication

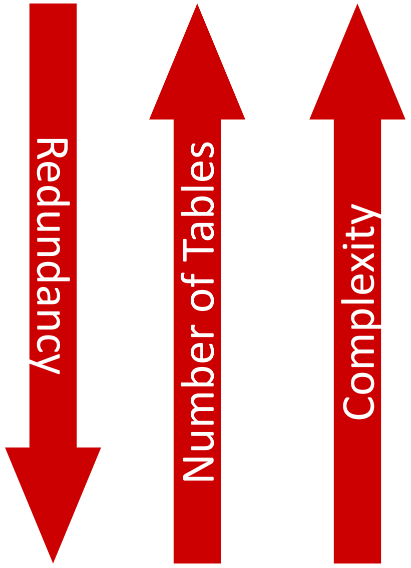
- Password based authentication is widely used, but is susceptible to sniffing on a network
- **Challenge-response** systems avoid transmission of passwords
  - DB sends a (randomly generated) challenge string to user
  - User encrypts string and returns result.
  - DB verifies identity by decrypting result
  - Can use public-key encryption system by DB sending a message encrypted using user's public key, and user decrypting and sending the message back
- **Digital signatures** are used to verify authenticity of data
  - E.g. use private key (in reverse) to encrypt data, and anyone can verify authenticity by using public key (in reverse) to decrypt data. Only holder of private key could have created the encrypted data.
  - Digital signatures also help ensure **nonrepudiation:** sender cannot later claim to have not created the data

Definition: Normalization

- This is the process which allows you to analyze out redundant data within your database.
- This involves restructuring the tables to successively meeting higher forms of Normalization.
- A properly normalized database should have the following characteristics
  - Scalar values in each fields
  - Absence of redundancy.
  - Minimal use of null values.
  - Minimal loss of information.

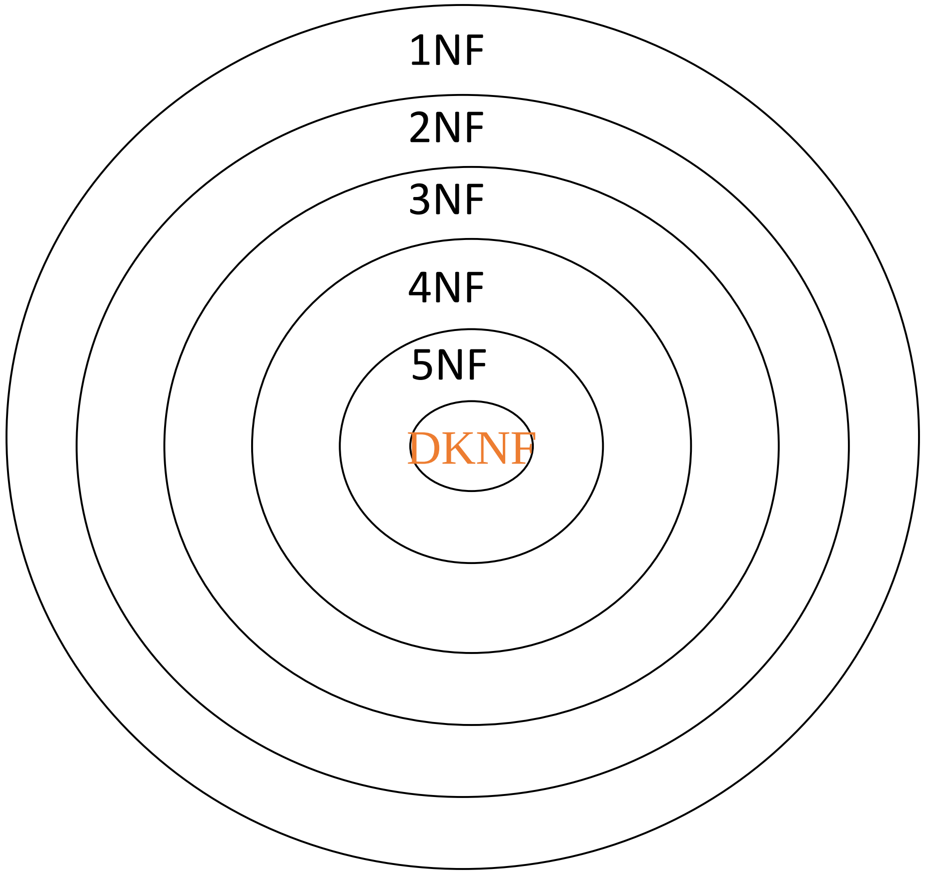
Levels of Normalization

- Levels of normalization based on the amount of redundancy in the database.
- Various levels of normalization are:
  - 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 (DKNF)



Most databases should be 3NF or BCNF in order to avoid the database anomalies.

Levels of Normalization



Each higher level is a subset of the lower level

First Normal Form (1NF)

A table is considered to be in 1NF if all the fields contain only scalar values (as opposed to list of values).

Example (Not 1NF)

ISBN	Title	AuName	AuPhone	PubName	PubPhone	Price
0-321-32132-1	Balloon	Sleepy, Snoopy, Grumpy	321-321-1111, 232-234-1234, 665-235-6532	Small House	714-000-0000	$34.00
0-55-123456-9	Main Street	Jones, Smith	123-333-3333, 654-223-3455	Small House	714-000-0000	$22.95
0-123-45678-0	Ulysses	Joyce	666-666-6666	Alpha Press	999-999-9999	$34.00
1-22-233700-0	Visual Basic	Roman	444-444-4444	Big House	123-456-7890	$25.00

Author and AuPhone columns are not scalar

1NF - Decomposition

1. Place all items that appear in the repeating group in a new table
2. Designate a primary key for each new table produced.
3. Duplicate in the new table the primary key of the table from which the repeating group was extracted or vice versa.

Example (1NF)

ISBN	Title	PubName	PubPhone	Price
0-321-32132-1	Balloon	Small House	714-000-0000	$34.00
0-55-123456-9	Main Street	Small House	714-000-0000	$22.95
0-123-45678-0	Ulysses	Alpha Press	999-999-9999	$34.00
1-22-233700-0	Visual Basic	Big House	123-456-7890	$25.00

ISBN	AuName	AuPhone
0-321-32132-1	Sleepy	321-321-1111
0-321-32132-1	Snoopy	232-234-1234
0-321-32132-1	Grumpy	665-235-6532
0-55-123456-9	Jones	123-333-3333
0-55-123456-9	Smith	654-223-3455
0-123-45678-0	Joyce	666-666-6666
1-22-233700-0	Roman	444-444-4444

Functional Dependencies

1. If one set of attributes in a table determines another set of attributes in the table, then the second set of attributes is said to be functionally dependent on the first set of attributes.

Example 1

ISBN	Title	Price
0-321-32132-1	Balloon	\$34.00
0-55-123456-9	Main Street	\$22.95
0-123-45678-0	Ulysses	\$34.00
1-22-233700-0	Visual Basic	\$25.00

Table Scheme: {ISBN, Title, Price}

Functional Dependencies: {ISBN} → {Title}

{ISBN} → {Price}

Functional Dependencies

Example 2

PubID	PubName	PubPhone
1	Big House	999-999-9999
2	Small House	123-456-7890
3	Alpha Press	111-111-1111

Table Scheme: {PubID, PubName, PubPhone}

Functional Dependencies: {PubId} → {PubPhone}

{PubId} → {PubName}

{PubName, PubPhone} → {PubID}

Example 3

AuID	AuName	AuPhone
1	Sleepy	321-321-1111
2	Snoopy	232-234-1234
3	Grumpy	665-235-6532
4	Jones	123-333-3333
5	Smith	654-223-3455
6	Joyce	666-666-6666
7	Roman	444-444-4444

Table Scheme: {AuID, AuName, AuPhone}

Functional Dependencies: {AuId} → {AuPhone}

{AuId} → {AuName}

{AuName, AuPhone} → {AuID}

FD – Example

Database to track reviews of papers submitted to an academic conference. Prospective authors submit papers for review and possible acceptance in the published conference proceedings. Details of the entities

- Author information includes a unique author number, a name, a mailing address, and a unique (optional) email address.
- Paper information includes the primary author, the paper number, the title, the abstract, and review status (pending, accepted,rejected)
- Reviewer information includes the reviewer number, the name, the mailing address, and a unique (optional) email address
- A completed review includes the reviewer number, the date, the paper number, comments to the authors, comments to the program chairperson, and ratings (overall, originality, correctness, style, clarity)

FD – Example

Functional Dependencies

- AuthNo → AuthName, AuthEmail, AuthAddress
- AuthEmail → AuthNo
- PaperNo → Primary-AuthNo, Title, Abstract, Status
- RevNo → RevName, RevEmail, RevAddress
- RevEmail → RevNo
- RevNo, PaperNo → AuthComm, Prog-Comm, Date, Rating1, Rating2, Rating3, Rating4, Rating5

Second Normal Form (2NF)

For a table to be in 2NF, there are two requirements

- The database is in first normal form
- All **nonkey** attributes in the table must be functionally dependent on the entire primary key

Note: *Remember that we are dealing with non-key attributes*

Example 1 (Not 2NF)

Scheme → {Title, PubId, AuId, Price, AuAddress}

1. **Key → {Title, PubId, AuId}**
2. **{Title, PubId, AuID} → {Price}**
3. **{AuID} → {AuAddress}**
4. **AuAddress does not belong to a key**
5. **AuAddress functionally depends on AuId which is a subset of a key**

Second Normal Form (2NF)

Example 2 (Not 2NF)

Scheme → {City, Street, HouseNumber, HouseColor, CityPopulation}

1. key → {City, Street, HouseNumber}
2. {City, Street, HouseNumber} → {HouseColor}
3. {City} → {CityPopulation}
4. CityPopulation does not belong to any key.
5. CityPopulation is functionally dependent on the City which is a proper subset of the key

Example 3 (Not 2NF)

Scheme → {studio, movie, budget, studio_city}

1. Key → {studio, movie}
2. {studio, movie} → {budget}
3. {studio} → {studio_city}
4. studio_city is not a part of a key
5. studio_city functionally depends on studio which is a proper subset of the key

2NF - Decomposition

1. If a data item is fully functionally dependent on only a part of the primary key, move that data item and that part of the primary key to a new table.
2. If other data items are functionally dependent on the same part of the key, place them in the new table also
3. Make the partial primary key copied from the original table the primary key for the new table. Place all items that appear in the repeating group in a new table

Example 1 (Convert to 2NF)

Old Scheme → {Title, PubId, AuId, Price, AuAddress}

New Scheme → {Title, PubId, AuId, Price}

New Scheme → {AuId, AuAddress}

2NF - Decomposition

Example 2 (Convert to 2NF)

Old Scheme → {Studio, Movie, Budget, StudioCity}

New Scheme → {Movie, Studio, Budget}

New Scheme → {Studio, City}

Example 3 (Convert to 2NF)

Old Scheme → {City, Street, HouseNumber, HouseColor, CityPopulation}

New Scheme → {City, Street, HouseNumber, HouseColor}

New Scheme → {City, CityPopulation}

Third Normal Form (3NF)

This form dictates that all **non-key** attributes of a table must be functionally dependent on a candidate key i.e. there can be no interdependencies among non-key attributes.

For a table to be in 3NF, there are two requirements

- The table should be second normal form
- No attribute is transitively dependent on the primary key

Example (Not in 3NF)

Scheme → {Title, PubID, PageCount, Price }

1. **Key → {Title, PubId}**
2. **{Title, PubId} → {PageCount}**
3. **{PageCount} → {Price}**
4. **Both Price and PageCount depend on a key hence 2NF**
5. **Transitively {Title, PubID} → {Price} hence not in 3NF**

Third Normal Form (3NF)

Example 2 (Not in 3NF)

Scheme → {<u>Studio</u>, StudioCity, CityTemp}

1. Primary Key → {Studio}
2. {Studio} → {StudioCity}
3. {StudioCity} → {CityTemp}
4. {Studio} → {CityTemp}
5. Both StudioCity and CityTemp depend on the entire key hence 2NF
6. CityTemp transitively depends on Studio hence violates 3NF

Example 3 (Not in 3NF)

Scheme → {BuildingID, Contractor, Fee}

1. Primary Key → {BuildingID}
2. {BuildingID} → {Contractor}
3. {Contractor} → {Fee}
4. {BuildingID} → {Fee}
5. Fee transitively depends on the BuildingID
6. Both Contractor and Fee depend on the entire key hence 2NF

BuildingID	Contractor	Fee
100	Randolph	1200
150	Ingersoll	1100
200	Randolph	1200
250	Pitkin	1100
300	Randolph	1200

3NF - Decomposition

1. Move all items involved in transitive dependencies to a new entity.
2. Identify a primary key for the new entity.
3. Place the primary key for the new entity as a foreign key on the original entity.

Example 1 (Convert to 3NF)

Old Scheme → {Title, PubID, PageCount, Price }

New Scheme → {PubID, PageCount, Price}

New Scheme → {Title, PubID, PageCount}

3NF - Decomposition

Example 2 (Convert to 3NF)

Old Scheme → {<u>Studio</u>, StudioCity, CityTemp}

New Scheme → {<u>Studio</u>, StudioCity}

New Scheme → {<u>StudioCity</u>, CityTemp}

Example 3 (Convert to 3NF)

Old Scheme → {BuildingID, Contractor, Fee}

New Scheme → {BuildingID, Contractor}

New Scheme → {Contractor, Fee}

BuildingID	Contractor
100	Randolph
150	Ingersoll
200	Randolph
250	Pitkin
300	Randolph

Contractor	Fee
Randolph	1200
Ingersoll	1100
Pitkin	1100

Boyce-Codd Normal Form (BCNF)

- BCNF does not allow dependencies between attributes that belong to candidate keys.
- BCNF is a refinement of the third normal form in which it drops the restriction of a non-key attribute from the 3rd normal form.
- Third normal form and BCNF are not same if the following conditions are true:
  - The table has two or more candidate keys
  - At least two of the candidate keys are composed of more than one attribute
  - The keys are not disjoint i.e. The composite candidate keys share some attributes

Example 1 - Address (Not in BCNF)

Scheme → {City, Street, ZipCode }

1. **Key1 → {City, Street }**
2. **Key2 → {ZipCode, Street}**
3. **No non-key attribute hence 3NF**
4. **{City, Street} → {ZipCode}**
5. **{ZipCode} → {City}**
6. **Dependency between attributes belonging to a key**

Boyce Codd Normal Form (BCNF)

Example 2 - Movie (Not in BCNF)

Scheme → {MovieTitle, MovieID, PersonName, Role, Payment }

1. Key1 → {MovieTitle, PersonName}
2. Key2 → {MovieID, PersonName}
3. Both role and payment functionally depend on both candidate keys thus 3NF
4. {MovieID} → {MovieTitle}
5. Dependency between MovieID & MovieTitle Violates BCNF

Example 3 - Consulting (Not in BCNF)

Scheme → {Client, Problem, Consultant}

1. Key1 → {Client, Problem}
2. Key2 → {Client, Consultant}
3. No non-key attribute hence 3NF
4. {Client, Problem} → {Consultant}
5. {Client, Consultant} → {Problem}
6. Dependency between attributess belonging to keys violates BCNF

BCNF - Decomposition

1. Place the two candidate primary keys in separate entities
2. Place each of the remaining data items in one of the resulting entities according to its dependency on the primary key.

Example 1 (Convert to BCNF)

Old Scheme → {City, Street, ZipCode }

New Scheme1 → {ZipCode, Street}

New Scheme2 → {City, Street}

- Loss of relation {ZipCode} → {City}

Alternate New Scheme1 → {ZipCode, Street }

Alternate New Scheme2 → {ZipCode, City}

Decomposition – Loss of Information

1. If decomposition does not cause any loss of information it is called a **lossless** decomposition.
2. If a decomposition does not cause any dependencies to be lost it is called a **dependency-preserving** decomposition.
3. Any table scheme can be decomposed in a lossless way into a collection of smaller schemas that are in BCNF form. However the dependency preservation is not guaranteed.
4. Any table can be decomposed in a lossless way into 3rd normal form that also preserves the dependencies.
   - 3NF may be better than BCNF in some cases

Use your own judgment when decomposing schemas

BCNF - Decomposition

Example 2 (Convert to BCNF)

Old Scheme → {MovieTitle, MovieID, PersonName, Role, Payment }

New Scheme → {<u>MovieID, PersonName</u>, Role, Payment}

New Scheme → {<u>MovieTitle, PersonName</u>}

- Loss of relation {MovieID} → {MovieTitle}

New Scheme → {<u>MovieID, PersonName</u>, Role, Payment}

New Scheme → {<u>MovieID, MovieTitle</u>}

- We got the {MovieID} → {MovieTitle} relationship back

Example 3 (Convert to BCNF)

Old Scheme → {Client, Problem, Consultant}

New Scheme → {Client, Consultant}

New Scheme → {Client, Problem}

Multi valued Dependencies (MVD)

- MVD occurs when there are two columns A&B and if for a given A there can be multiple values of B then we say that MVD exist between A&B. If there are two of more such unrelated MVD relationship in a table, it violets the 4NF.

table:Project

Name	ProjectNo	Hobby
A	P1	Football
A	P2	Football
B	P3	Gulf
B	P3	Tennis

Name->-> ProjectNo

Name->-> Hobby

Fourth Normal Form (4NF)

- Fourth normal form eliminates independent many-to-one relationships between columns.
- To be in Fourth Normal Form,
  - a relation must first be in Boyce-Codd Normal Form.
  - a given relation may not contain more than one multi-valued attribute.

Example (Not in 4NF)

Scheme → {MovieName, ScreeningCity, Genre)

Primary Key: {MovieName, ScreeningCity, Genre)

1. All columns are a part of the only candidate key, hence BCNF
2. Many Movies can have the same Genre
3. Many Cities can have the same movie
4. Violates 4NF

Movie	ScreeningCity	Genre
Hard Code	Los Angles	Comedy
Hard Code	New York	Comedy
Bill Durham	Santa Cruz	Drama
Bill Durham	Durham	Drama
The Code Warrier	New York	Horror

Fourth Normal Form (4NF)

Example 2 (Not in 4NF)

Scheme → {Manager, Child, Employee}

1. Primary Key → {Manager, Child, Employee}
2. Each manager can have more than one child
3. Each manager can supervise more than one employee
4. 4NF Violated

Manager	Child	Employee
Jim	Beth	Alice
Mary	Bob	Jane
Mary	NULL	Adam

Example 3 (Not in 4NF)

Scheme → {Employee, Skill, Language}

1. Primary Key → {Employee, Skill, Language }
2. Each employee can speak multiple languages
3. Each employee can have multiple skills
4. Thus violates 4NF

Employee	Skill	Language
1234	Cooking	French
1234	Cooking	German
1453	Carpentry	Spanish
1453	Cooking	Spanish
2345	Cooking	Spanish

4NF - Decomposition

1. Move the two multi-valued relations to separate tables
2. Identify a primary key for each of the new entity.

Example 1 (Convert to 3NF)

Old Scheme → {MovieName, ScreeningCity, Genre}

New Scheme → {MovieName, ScreeningCity}

New Scheme → {MovieName, Genre}

Movie	Genre
Hard Code	Comedy
Bill Durham	Drama
The Code Warrier	Horror

Movie	ScreeningCity
Hard Code	Los Angles
Hard Code	New York
Bill Durham	Santa Cruz
Bill Durham	Durham
The Code Warrier	New York

4NF - Decomposition

Example 2 (Convert to 4NF)

Old Scheme → {Manager, Child, Employee}

New Scheme → {Manager, Child}

New Scheme → {Manager, Employee}

Manager	Child
Jim	Beth
Mary	Bob

Manager	Employee
Jim	Alice
Mary	Jane
Mary	Adam

Example 3 (Convert to 4NF)

Old Scheme → {Employee, Skill, Language}

New Scheme → {Employee, Skill}

New Scheme → {Employee, Language}

Employee	Skill
1234	Cooking
1453	Carpentry
1453	Cooking
2345	Cooking

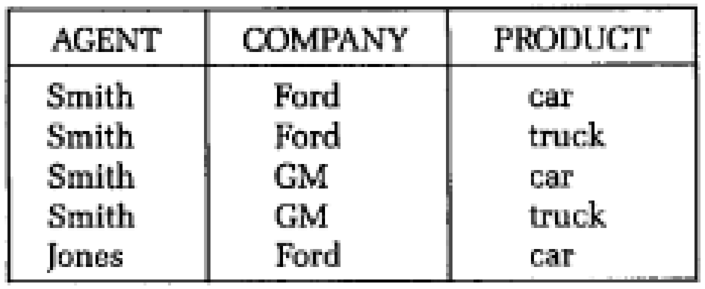
Employee	Language
1234	French
1234	German
1453	Spanish
2345	Spanish

Fifth Normal Form

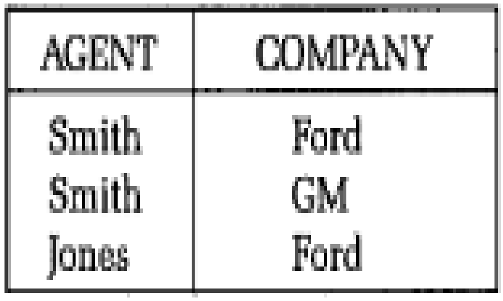
- A relation is in 5NF if every join dependency in the relation is implied by the keys of the relation
- *Implies that relations that have been decomposed in previous normal forms can be recombined via natural joins to recreate the original relation.*

Fifth Normal Form (5NF)

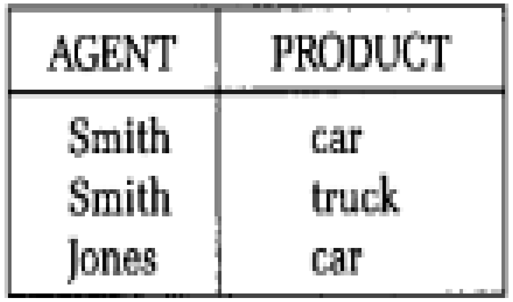
- Fifth normal form is satisfied when all tables are broken into as many tables as possible in order to avoid redundancy. Once it is in fifth normal form it cannot be broken into smaller relations without changing the facts or the meaning.



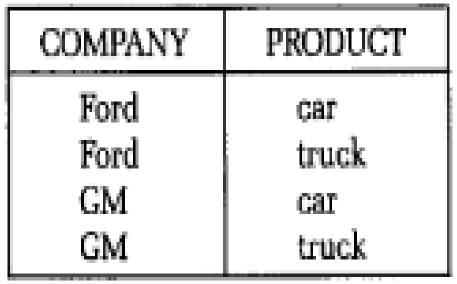
AGENT	COMPANY	PRODUCT
Smith	Ford	car
Smith	Ford	truck
Smith	GM	car
Smith	GM	truck
Jones	Ford	car



AGENT	COMPANY
Smith	Ford
Smith	GM
Jones	Ford



AGENT	PRODUCT
Smith	car
Smith	truck
Jones	car



COMPANY	PRODUCT
Ford	car
Ford	truck
GM	car
GM	truck

Domain Key Normal Form (DKNF)

- The relation is in DKNF when there can be no insertion or deletion anomalies in the database.
- A relation is in DKNF when insertion or delete anomalies are not present in the database. Domain-Key Normal Form is the highest form of Normalization. The reason is that the insertion and updation anomalies are removed. The constraints are verified by the domain and key constraints.
- A table is in Domain-Key normal form only if it is in 4NF, 3NF and other normal forms. It is based on constraints:

Domain Constraint

Values of an attribute had some set of values, for example, EmployeeID should be four digits long:

EmpID	EmpName	EmpAge
0921	Hari	25
0922	Geeta	24

Key Constraint

An attribute or its combination is a candidate key

General Constraint

Predicate on the set of all relations.

Every constraint should be a logical sequence of the domain constraints and key constraints applied to the relation. The practical utility of DKNF is less.

Denormalization

- Denormalization is the process of adding redundant columns and tables to the database in order to improve performance.

E.g. if the phone number of the instructor of a course is often needed, the column PHONE can be added to the table COURSES.

Then the join between COURSES and INSTRUCTORS can often be avoided, because the required instructor information (PHONE) is stored redundantly in the same row as the course information.

- Since there is a separate table INSTRUCTORS (which contains the phone number, too), insertion and deletion anomalies are avoided.

But there will be update anomalies (changing a single phone number requires updating many rows).

Thus, one must pay for the performance gain with a more complicated application logic, the need for triggers, and some remaining insecurity (will all copies always agree?).

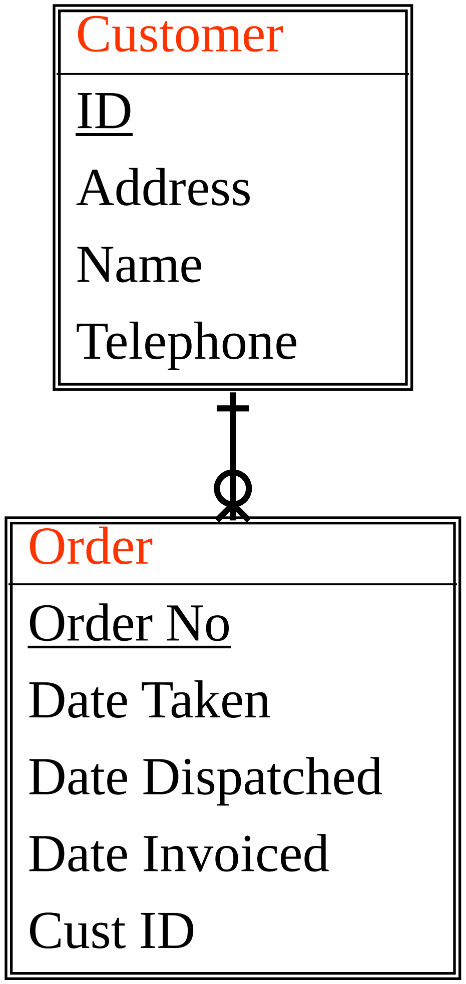
- If the tables COURSES and INSTRUCTORS are small, it is certainly a bad decision to violate BCNF here, since performance is anyway no problem.
  - Koletzke/Dorsey state that if your tables have less than 100000 rows, you need no denormalization. Of course, it also depends on the number of queries per second.
- Too much denormalization can make a database nearly unmodifiable.

The requirements often change, so one must anticipate that the DB application system will need changes.

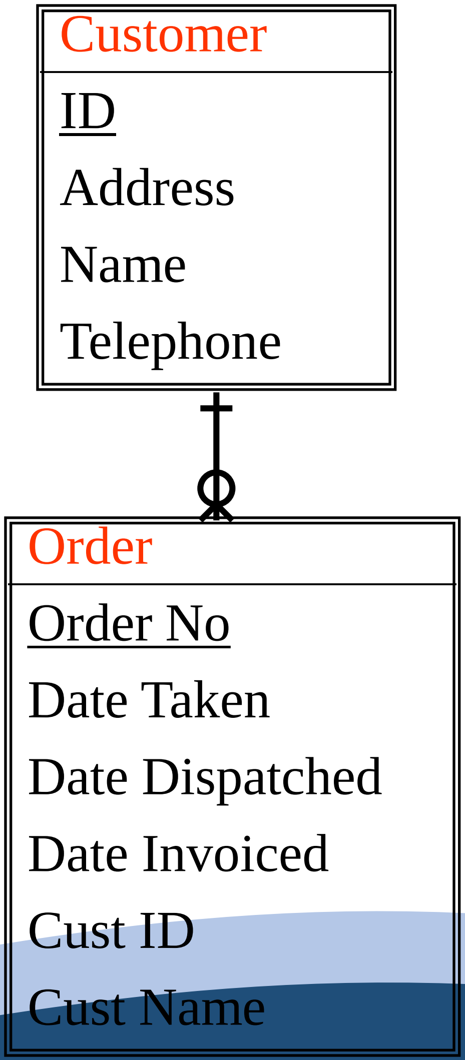
- In the above example, denormalization helped to avoid joins. Denormalization also includes creating tables or columns which hold aggregated values. In this case, formally no normal form is violated, but the information is of course redundant.
- E.g. suppose one stores invoices to a customer, and payments from a customer. One could e.g. store in the CUSTOMERS table the current balance. This is redundant, because it can be computed as the sum of all payments minus the sum of all invoices.

Downward Denormalization

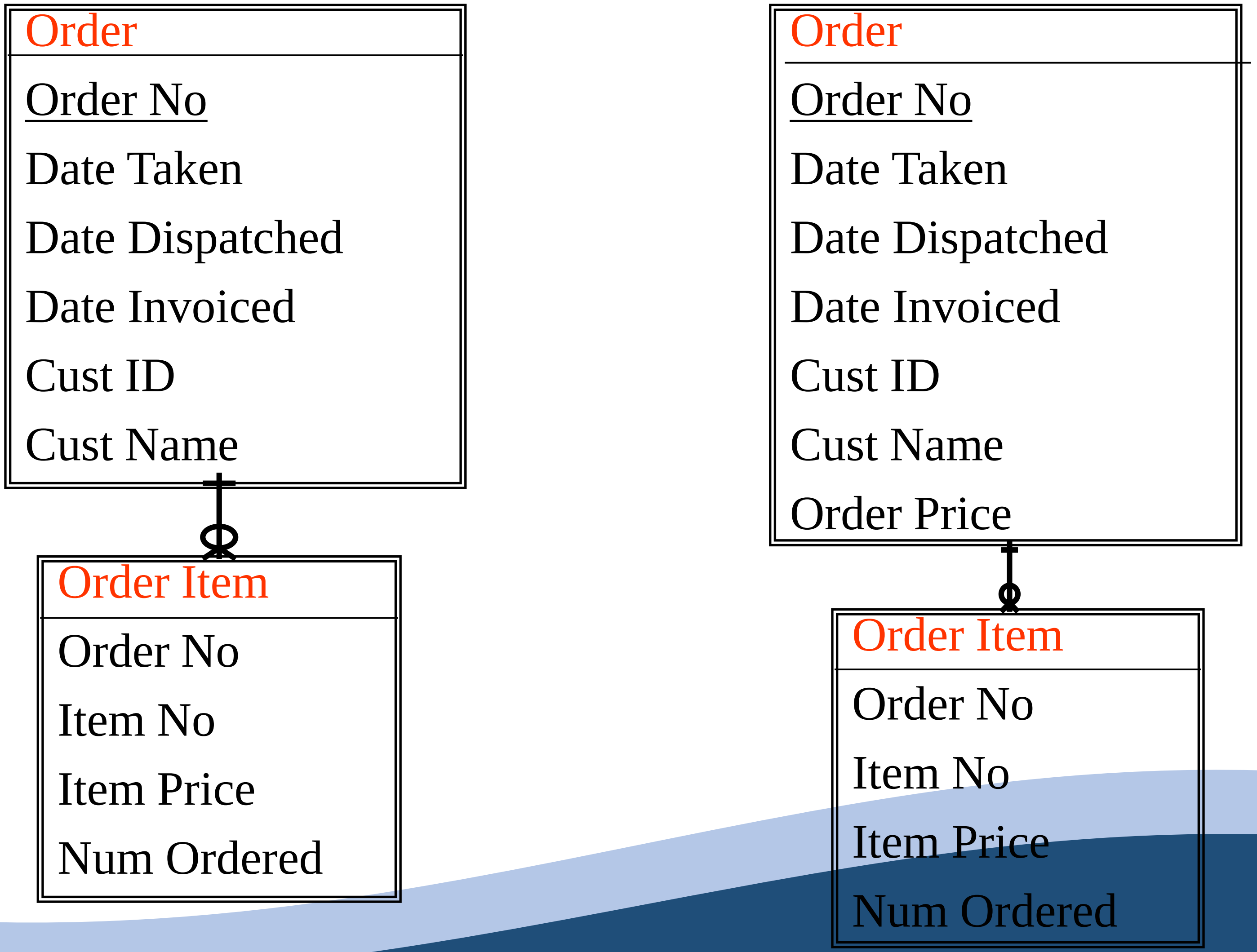
Before:



After:



Upward Denormalization



Order
Order No
Date Taken
Date Dispatched
Date Invoiced
Cust ID
Cust Name
Order Item
Order No
Item No
Item Price
Num Ordered
Order
Order No
Date Taken
Date Dispatched
Date Invoiced
Cust ID
Cust Name
Order Price
Order Item
Order No
Item No
Item Price
Num Ordered