



SCTR's

Pune Institute of Computer Technology, Pune - 411043.

Department of Electronics & Telecommunication



Unit II - Relational Database Design

Mr. Sandeep L Dhende

Assistant Professor, Dept. of E&TC, PICT, Pune



Outline

- Relational Model: Basic concepts, Attributes and Domains
- Relational Integrity: Domain, Referential Integrities
- CODD's Rules
- Features of Good Relational Design
- Atomic Domains and First Normal Form
- Decomposition Using Functional Dependencies
- Functional Dependency Theory
- Normalization



Relational Model

- Relational model stores data in the form of tables. This concept proposed by Dr. E. F. Codd, a researcher of IBM in the year 1960s.
- The relational model consists of three major components:
 - The set of relations and set of
 - Integrity rules
 - The operations
- A relational model database is defined as a database that allows you to group its data items into one or more independent tables that can be related to one another by using fields common to each related table.



Basic Structure

- Formally, given sets D_1, D_2, \dots, D_n a **relation** r is a subset of $D_1 \times D_2 \times \dots \times D_n$
Thus a relation is a set of n-tuples (a_1, a_2, \dots, a_n) where each $a_i \in D_i$

- Example:if

$customer-name = \{Jones, Smith, Curry\}$

$customer-street = \{Main, North, Park\}$

$customer-city = \{Harrison, Rye, Pittsfield\}$

Then $r = \{(Jones, Main, Harrison),$

$(Smith, North, Rye),$

$(Curry, Park, Pittsfield)\}$

is a relation over $customer-name \times customer-street \times customer-city$



Characteristics of Relational Database

- Relational database systems have the following characteristics:
 - The whole data is conceptually represented as an orderly arrangement of data into rows and columns, called a **relation or table**.
 - All values are scalar. That is, at any given row/column position in the relation there is one and only one value.
 - All operations are performed on an entire relation and result is an entire relation, a concept known as closure.



Basic Terminology used in Relational Model

- The figure shows a relation for relational database

Attributes	Emp_Code	Name	Year
Tuples	21130	Amar Jain	1
	30745	Kuldeep	3
	41894	Manoj	2
	51207	Rita bajaj	6

- A single row of a table, which contains a single record for that relation is called a **tuple**. Actually, each row is an n-tuple, but the "n-" is usually dropped.



Basic Terminology used in Relational Model

- **Relation instance** – A finite set of tuples in the relational database system represents relation instance. Relation instances do not have duplicate tuples.
- **Relation schema** – A relation schema describes the relation name (table name), attributes, and their names.

A_1, A_2, \dots, A_n are *attributes*

$R = (A_1, A_2, \dots, A_n)$ is a *relation schema*

E.g. *Customer-schema* = (*customer-name, customer-street, customer-city*)

- **Cardinality of a relation:** The number of tuples in a relation determines its cardinality. In this case, the relation has a cardinality of 4.
- **Degree of a relation:** Each column in the tuple is called an attribute. The number of attributes in a relation determines its degree. The relation in figure has a degree of 3.
- **Domains:** A domain definition specifies the kind of data represented by the attribute.



Basic Terminology used in Relational Model

- Keys are very important part of Relational database. They are used to establish and identify relation between tables. They also ensure that each record within a table can be uniquely identified by combination of one or more fields within a table.
- **Super Key –**
Super Key is defined as a set of attributes within a table that uniquely identifies each record within a table. Super Key is a superset of Candidate key.
- **Candidate Key –**
Candidate keys are defined as the set of fields from which primary key can be selected. It is an attribute or set of attribute that can act as a primary key for a table to uniquely identify each record in that table.
- **Primary Key**
Primary key is a candidate key that is most appropriate to become main key of the table. It is a key that uniquely identify each record in a table.




Basic Terminology used in Relational Model

■ Primary Key

Primary key is a candidate key that is most appropriate to become main key of the table. It is a key that uniquely identify each record in a table.

Primary Key

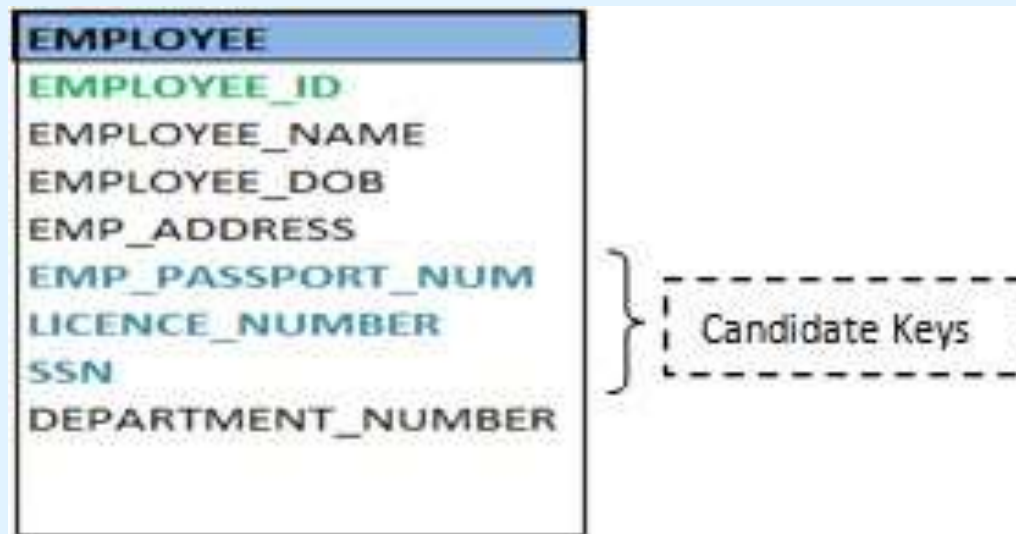


s_id	S_name	age	course	address



Basic Terminology used in Relational Model

- **Candidate Key -**
- **Example** of employee table, EMPLOYEE_ID is best suited for primary key as its from his own employer. Rest of the attributes like passport number, SSN, license Number etc. are considered as candidate key.





Basic Terminology used in Relational Model

■ Composite Key

Key that consist of two or more attributes that uniquely identify an entity occurrence is called **Composite key**. But any attribute that makes up the **Composite key** is not a simple key in its own.

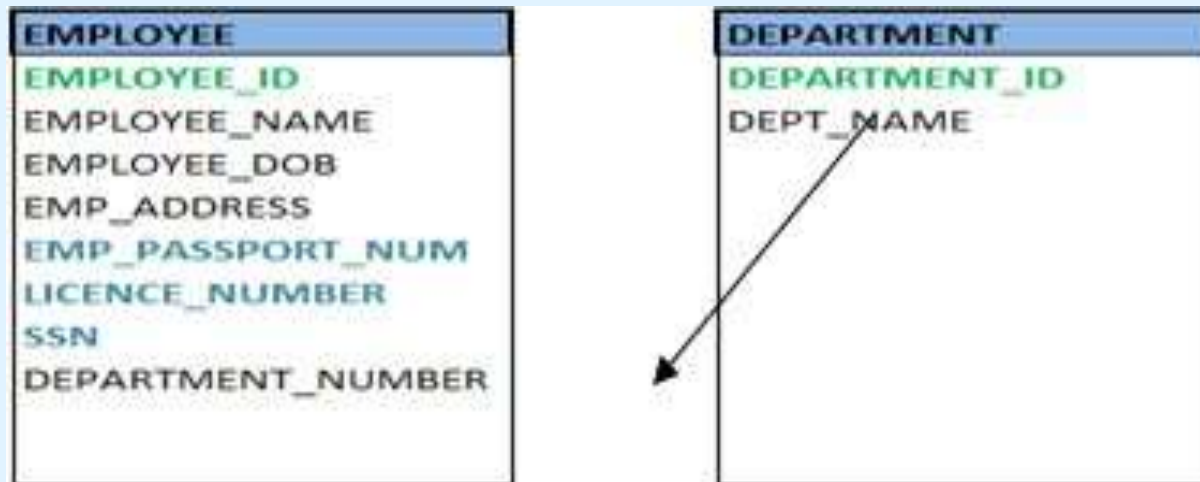




Basic Terminology used in Relational Model

■ Foreign key -

A **foreign key** is a field (or collection of fields) in one table that uniquely identifies a row of another table or the same table. In simpler words, the **foreign key** is defined in a second table, but it refers to the primary **key** in the first table.





Basic Terminology used in Relational Model

■ Secondary or Alternative key

The candidate key which are not selected for primary key are known as secondary keys or alternative keys

■ Non-key Attribute

Non-key attributes are attributes other than **candidate key** attributes in a table.

■ Non-prime Attribute

Non-prime Attributes are attributes other than **Primary attribute**.



Basic Terminology used in Relational Model

- **Constraints-** Every relation has some conditions that must hold for it to be a valid relation. These conditions are called **Relational Integrity Constraints**. There are three main integrity constraints –
 - Key constraints
 - Domain constraints
 - Referential integrity constraints
- **Key Constraints** - There must be at least one minimal subset of attributes in the relation, which can identify a tuple uniquely. This minimal subset of attributes is called **key** for that relation. If there are more than one such minimal subsets, these are called *candidate keys*. Key constraints force that –
 - in a relation with a key attribute, no two tuples can have identical values for key attributes.
 - a key attribute can not have NULL values.
- Key constraints are also referred to as Entity Constraints.



Basic Terminology used in Relational Model

- **Domain Constraints** - Attributes have specific values in real-world scenario. For example, age can only be a positive integer. The same constraints have been tried to employ on the attributes of a relation. Every attribute is bound to have a specific range of values. For example, age cannot be less than zero and telephone numbers cannot contain a digit outside 0-9.
- **Referential integrity Constraints** - Referential integrity constraints work on the concept of Foreign Keys. A foreign key is a key attribute of a relation that can be referred in other relation.
- Referential integrity constraint states that if a relation refers to a key attribute of a different or same relation, then that key element must exist.
- A foreign key must have a matching primary key or it must be null.



Basic Terminology used in Relational Model

- **Example:** Customer table contains the **information** about the customers with CNO as the primary key. The Customer_Loan table stores the information about CNO, LNO and AMOUNT. It has the primary key combination of CNO and LNO. Here, CNO also acts as the foreign key and refers to CNO of Customer table.

Customer Table			
CNO	NAME	ADDRESS	CITY
C1	Rahal	Thapar campus	Patlala
C2	Ruhi	Tagore Nagar	Jalandhar
C3	Chachat	Dharampura	Qadian
C4	Pooja	GNDU	Amritsar

Customer_Loan Table		
CNO	LNO	AMOUNT
C1	L1	10000
C2	L1	10000
C3	L2	15000
C3	L3	25000
C4	L4	35000

Relationat Model of Custmor-Loan database



Codd's 12 Rules

- Dr. Edgar F. Codd, after his extensive research on the Relational Model of database systems, came up with twelve rules. These rules can be applied on any database system that manages stored data using only its relational capabilities. This is a foundation rule, which acts as a base for all the other rules.
- **Rule 1: Information Rule** - The data stored in a database, may it be user data or metadata, must be a value of some table cell. Everything in a database must be stored in a table format.
- **Rule 2: Guaranteed Access Rule** - Every single data element (value) is guaranteed to be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column value). Such as pointers, can be used to access data.



Codd's 12 Rules

- **Rule 3: Systematic Treatment of NULL Values** - The NULL values in a database must be given a systematic and uniform treatment. This is a very important rule because a NULL can be interpreted as one the following – data is missing, data is not known, or data is not applicable.
- **Rule 4: Active Online Catalog** - The structure description of the entire database must be stored in an online catalog, known as **data dictionary**, which can be accessed by authorized users. Users can use the same query language to access the catalog which they use to access the database itself.
- **Rule 5: Comprehensive Data Sub-Language Rule** - A database can only be accessed using a language having linear syntax that supports data definition, data manipulation, and transaction management operations. This language can be used directly or by means of some application. If the database allows access to data without any help of this language, then it is considered as a violation.



Codd's 12 Rules

- **Rule 6: View Updating Rule** - All the views of a database, which can theoretically be updated, must also be updatable by the system.
- **Rule 7: High-Level Insert, Update, and Delete Rule** - A database must support high-level insertion, updation, and deletion. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.
- **Rule 8: Physical Data Independence** - The data stored in a database must be independent of the applications that access the database. Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.



Codd's 12 Rules

- **Rule 9: Logical Data Independence** - The logical data in a database must be independent of its user's view (application). Any change in logical data must not affect the applications using it. For example, if two tables are merged or one is split into two different tables, there should be no impact or change on the user application. This is one of the most difficult rule to apply
- **Rule 10: Integrity Independence** - A database must be independent of the application that uses it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes a database independent of the front-end application and its interface.



Codd's 12 Rules

- **Rule 11: Distribution Independence** - The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only. This rule has been regarded as the foundation of distributed database systems.
- **Rule 12: Non-Subversion Rule** - If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.



Schemas

- The *instructor* and *department* tables

ID	name	dept_name	salary
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table

dept_name	building	budget
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

(b) The *department* table



Combine Schemas

- Suppose we combine *instructor* and *department* into *inst_dept*
 - (No connection to relationship set *inst_dept*)
- Result is possible repetition of information

ID	name	salary	dept_name	building	budget
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000



A Combined Schema Without Repetition

- Consider combining relations
 - *sec_class(sec_id, building, room_number)* and
 - *section(course_id, sec_id, semester, year)*into one relation
 - *section(course_id, sec_id, semester, year, building, room_number)*
- No repetition in this case

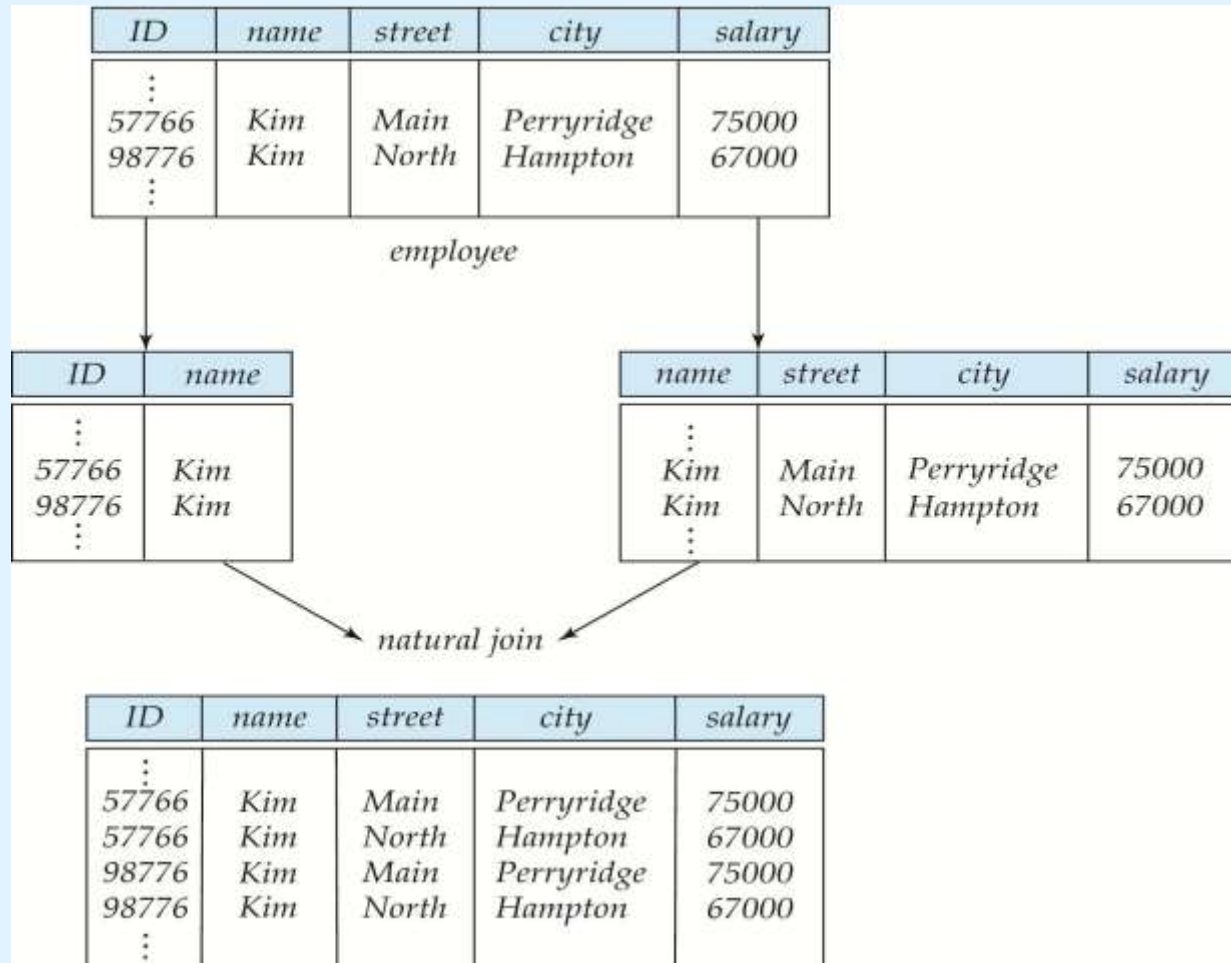


What About Smaller Schemas?

- Suppose we had started with *inst_dept*. How would we know to split up (**decompose**) it into *instructor* and *department*?
- Write a rule “if there were a schema (*dept_name*, *building*, *budget*), then *dept_name* would be a candidate key”
- Denote as a **functional dependency**:
$$dept_name \rightarrow building, budget$$
- In *inst_dept*, because *dept_name* is not a candidate key, the building and budget of a department may have to be repeated.
 - This indicates the need to decompose *inst_dept*
- Not all decompositions are good. Suppose we decompose *employee*(*ID*, *name*, *street*, *city*, *salary*) into
employee1 (*ID*, *name*)
employee2 (*name*, *street*, *city*, *salary*)
- The next slide shows how we lose information -- we cannot reconstruct the original *employee* relation -- and so, this is a **lossy decomposition**.



A Lossy Decomposition





A Lossy Decomposition

Roll	Name	Faculty_id
32301	A	35502
32302	B	35503

Faculty_id	Faculty_Name
35502	Anand
35503	Bose

■ Cartesian Product

Roll	Name	Faculty_id	Faculty_id	Faculty_Name
32301	A	35502	35502	Anand
32301	A	35502	35503	Bose
32302	B	35503	35502	Anand
32302	B	35503	35503	Bose



A Lossy Decomposition

Roll	Name	Faculty_id
32301	A	35502
32302	B	35503

Faculty_id	Faculty_Name
35502	Anand
35503	Bose

■ Natural Join

Roll	Name	Faculty_id	Faculty_id	Faculty_Name
32301	A	35502	35502	Anand
32301	A	35502	35503	Bose
32302	B	35503	35502	Anand
32302	B	35503	35503	Bose



A Lossy Decomposition

Roll	Name	Faculty_id
32301	A	35502
32302	B	35503

Faculty_id	Faculty_Name
35502	Anand
35503	Bose

■ Natural Join

Roll	Name	Faculty_id	Faculty_Name
32301	A	35502	Anand
32302	B	35503	Bose



Example of Lossless-Join Decomposition

■ Lossless join decomposition

■ Decomposition of $R = (A, B, C)$

$$R_1 = (A, B) \quad R_2 = (B, C)$$

A	B	C
α	1	A
β	2	B

 r

A	B
α	1
β	2

 $\Pi_{A,B}(r)$

B	C
1	A
2	B

 $\Pi_{B,C}(r)$

$$\Pi_{A,B}(r) \bowtie \Pi_{B,C}(r)$$

A	B	C
α	1	A
β	2	B



Functional Dependencies

- Constraints on the set of legal relations.
- Require that the value for a certain set of attributes determines uniquely the value for another set of attributes.
- A functional dependency is a generalization of the notion of a *key*.



Functional Dependencies (Cont.)

- Let R be a relation schema

$$\alpha \subseteq R \text{ and } \beta \subseteq R$$

- The **functional dependency**

$$\alpha \rightarrow \beta$$

holds on R if and only if for any legal relations $r(R)$, whenever any two tuples t_1 and t_2 of r agree on the attributes α , they also agree on the attributes β . That is,

$$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$$

- Example: Consider $r(A, B)$ with the following instance of r

1	4
1	5
3	7

- On this instance, $A \rightarrow B$ does NOT hold, but $B \rightarrow A$ does hold.



Functional Dependencies (Cont.)

- K is a superkey for relation schema R if and only if $K \rightarrow R$
- K is a candidate key for R if and only if
 - $K \rightarrow R$, and
 - for $\alpha \subset K$, $\alpha \not\rightarrow R$
- Functional dependencies allow us to express constraints that cannot be expressed using superkeys. Consider the schema:

inst_dept (ID, name, salary, dept_name, building, budget).

We expect these functional dependencies to hold:

dept_name \rightarrow *building*

and *ID* \rightarrow *building*

but would not expect the following to hold:

dept_name \rightarrow *salary*



Use of Functional Dependencies

- We use functional dependencies to:
 - test relations to see if they are legal under a given set of functional dependencies.
 - ▶ If a relation r is legal under a set F of functional dependencies, we say that r **satisfies** F
 - specify constraints on the set of legal relations
 - ▶ We say that F **holds on** R if all legal relations on R satisfy the set of functional dependencies F
- Note: A specific instance of a relation schema may satisfy a functional dependency even if the functional dependency does not hold on all legal instances.
 - For example, a specific instance of *instructor* may, by chance, satisfy $name \rightarrow ID$.



Use of Functional Dependencies

- Let us consider instance of relation r of following figure, to see which functional dependencies are satisfied.
- Observe $A \rightarrow C$ is satisfied.
- The functional dependency $C \rightarrow A$ is not satisfied.

A	B	C	D
a1	b1	c1	d1
a1	b2	c1	d2
a2	b2	c2	d2
a2	b3	c2	d3
a3	b3	c2	d4



Types of Functional Dependencies

1. Single-valued Functional Dependency
2. Multi-valued Functional Dependency
3. Fully Functional Dependency
4. Partial Functional Dependency
5. Transitive Functional Dependency
6. Trivial Function Dependency
7. Non Trivial Functional dependency



Types of Functional Dependencies

1. Single-valued Functional Dependency

- A database is collection of interrelated data.
- The data is either single valued or multi-valued.
- Example: The student name is single valued fact. But the student mobile is a multi-valued attribute.
- A single valued functional dependency is hold by a relation when A is the primary key of a relation and B is some single valued attribute of the relation.

- $A \rightarrow B$

Roll_No	Name	Mobile
32301	Ram	9786458956
32302	Vijay	8794563215
32304	Kumar	9874563254
32305	Santosh	7089463245



Types of Functional Dependencies

2. Multi-valued Functional Dependency

- It means for a single value of A multiple values of B exist.
- It is denoted by $A \twoheadrightarrow B$

Roll_No	Dept_No	Course_No
1	D1	C1
1	D1	C2
2	D1	C1
2	D1	C2
3	D2	C2

- $\text{Roll_No} \twoheadrightarrow \text{Course_No}$
- $\text{Roll_No} \rightarrow \text{Dept_No}$



Types of Functional Dependencies

2. Multi-valued Functional Dependency

- A multi-valued dependency represents a dependency between attributes (Let A, B & C) in a relation r such that for each value of A there is a set of values of B and a set of values of C.
- However the set of values of B & C are independent of each other.
- Because of this dependency there is a lot of redundancy in a relation r.

E_No	P_No	Mobile
A	1	9874561235
A	2	7894568213
A	3	9874561235
B	1	8795641235
B	2	9854625355



Types of Functional Dependencies

3. Fully Functional Dependency

- The fully functional dependency occurs only in a relation with composite primary key.
- It occurs when one or more non key attributes are depending on all of the parts of the composite primary key.

Roll_No	Name	Mobile	Course_No	Course_Name	Credits	Grade
1	A	9763242658	C1	DM	3	A
1	A	9763242658	C2	MC	3	B
2	B	8974563215	C1	DM	3	B
2	B	8974563215	C2	MC	3	B
3	C	8974563215	C3	DC	3	A



Types of Functional Dependencies

- The attribute Grade is fully functional dependent on the primary key (Roll_No + Course_No) because both parts of primary key are needed to determine Grade.
- The Roll_No determines both the Name and Mobile of the student.
- The Course_No determines both the Course_Name and Credits of the course.



Types of Functional Dependencies

4. Partial Functional Dependency

- The partial functional dependency occurs only in a relation with composite primary key.
- It occurs when one or more non key attributes are depending on a part of the primary key.
- Here the determinant consists of key attributes, but not the entire primary key and determined consist of non-key attributes.

Roll_No	Name	Mobile	Course_No	Course_Name	Credits	Grade
1	A	9763242658	C1	DM	3	A
1	A	9763242658	C2	MC	3	B
2	B	8974563215	C1	DM	3	B
2	B	8974563215	C2	MC	3	B
3	C	8974563215	C3	DC	3	A



Types of Functional Dependencies

5. Transitive Functional Dependency

- Functional dependency is said to be transitive if it is indirectly formed using two functional dependencies.
- Example: In a relation $R(A, B, C)$, Functional dependency founds:
- $A \rightarrow B$ and $B \rightarrow C$ both the FD's are hold by relation, and id $B \rightarrow A$ does not hold.
- Then we say that $A \rightarrow C$ also holds by the relation.
- Example: $\text{Course_id} \rightarrow \text{Student_Name}$

$\text{Student_Name} \rightarrow \text{Student_Age}$

Then,

$\text{Course_id} \rightarrow \text{Student_Age}$



Types of Functional Dependencies

6. Trivial Functional Dependency

- A functional dependency is **trivial** if it is satisfied by all instances of a relation
 - Example:
 - ▶ $ID, name \rightarrow ID$
 - ▶ $name \rightarrow name$
 - In general, $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$



Types of Functional Dependencies

7. Non Trivial Functional Dependency

- It is inverse to trivial dependency.
- The functional dependency is said to be non trivial if non of the attributes of right hand side of FD are part of left hand side attributes.
- Example: The FD $A \rightarrow B$ is non trivial if values in B are not present in A i.e. (B is not subset of A).
- Further categories:
 - Complete Non trivial Functional dependency
Example: $\text{Roll_No} \rightarrow \text{Student_Name}$
 - Semi Non trivial Functional dependency
Example: $\{\text{Roll_No}, \text{Student_Name}\} \rightarrow \{\text{Student_Name}, \text{Mobile}\}$



Closure of a Set of Functional Dependencies

- Given a set F of functional dependencies, there are certain other functional dependencies that are logically implied by F .
 - For example: If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$
- The set of **all** functional dependencies logically implied by F is the **closure** of F .
- We denote the *closure* of F by **F^+** .
- F^+ is a superset of F .



Armstrong's Axioms

- To compute F^+ , we can use some rules of inference like Armstrong's axioms.
- The inference rule is an assertion which is used to derive new FDs.
- The new functional dependencies will be derived from previously defined functional dependency set.
- It is problematic to represent sets of all FDs for a relation initially.
- To make it easy, first define only basic FDs. Then apply set of inference rules on FDs to derive new set of FDs.
- The most basic rule is Armstrong's Axioms.
- There are other rules such as union, decomposition and pseudo transitivity which are derived from Axioms.



Armstrong's Axioms

- Armstrong's Axioms were developed by William W. Armstrong.
- There are set of axioms (or infer rules) which are used to infer all the FDs on a relational database.
- Consider relation R having three sets of attributes α , β and γ . Then the Armstrong's Axioms are:
 - if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ (**reflexivity**)
 - if $\alpha \rightarrow \beta$, then $\gamma \alpha \rightarrow \gamma \beta$ (**augmentation**)
 - if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ (**transitivity**)
- These rules are
 - **sound** (generate only functional dependencies that actually hold) or they do not generate any incorrect functional dependencies and
 - **complete** (generate all functional dependencies that hold).



Armstrong's Axioms

■ Additional rules:

- If $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta \gamma$ holds (**union**)
- If $\alpha \rightarrow \beta \gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds (**decomposition**)
- If $\alpha \rightarrow \beta$ holds and $\gamma \beta \rightarrow \delta$ holds, then $\alpha \gamma \rightarrow \delta$ holds (**pseudo transitivity**)

The above rules can be inferred from Armstrong's axioms.



Example

- $R = (A, B, C, G, H, I)$
 $F = \{ A \rightarrow B$
 $A \rightarrow C$
 $CG \rightarrow H$
 $CG \rightarrow I$
 $B \rightarrow H \}$
- Several members of F^+
 - $A \rightarrow H$
 - ▶ by transitivity from $A \rightarrow B$ and $B \rightarrow H$
 - $AG \rightarrow I$
 - ▶ by augmenting $A \rightarrow C$ with G , to get $AG \rightarrow CG$
 and then transitivity with $CG \rightarrow I$
 - $CG \rightarrow HI$
 - ▶ Since $CG \rightarrow H$ and $CG \rightarrow I$, then union rule implies that $CG \rightarrow HI$



Example of Attribute Set Closure

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B$
 $A \rightarrow C \quad CG \rightarrow H \quad CG \rightarrow I \quad B \rightarrow H\}$
- $(AG)^+$
 1. $result = AG$
 2. $result = ABCG$ $(A \rightarrow C \text{ and } A \rightarrow B)$
 3. $result = ABCGH$ $(CG \rightarrow H \text{ and } CG \subseteq AGBC)$
 4. $result = ABCGHI$ $(CG \rightarrow I \text{ and } CG \subseteq AGBCH)$



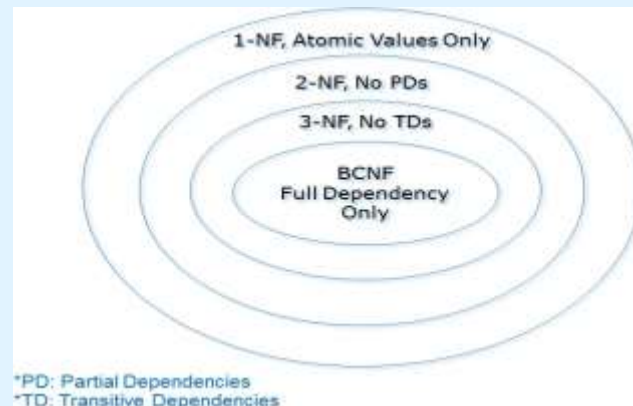
Normalization

- Dr E. F. Codd proposess normalization as integral part of relational model.
- The major aim of relational database design is to group attributes into relations to minimize data redundancy and reduce file storage space required by base relations.
- It is used to organize the tables in such manner that it should reduce redundancy and dependency of data.
- Normalization is the process of reducing/decomposing a relation into a set of small relations free from data redundancy and ensuring data integrity.
- Defined as a step-by-step process of decomposing a complex relation into a simple and stable data structure.
- How functional dependencies can be used to group attributes into relations that are in a known normal form.



Types of Normalization

- First Normal Form (1NF)
 - Second Normal Form (2NF)
 - Third Normal Form (3NF)
 - Fourth Normal Form (4NF)
 - Boyce-Codd Normal Form (BCNF)
-
- Each normal form is an improvement over the earlier form.
 - A higher normal form relation is a subset of lower normal form.





Need of Normalization

- In the database management process without normalization, it is not easy to manage database operations like insertion, deletion and modification without facing data loss problem.
- If database is not normalized then insertion, deletion and updation anomalies occurs frequently.
- **Anomalies??**
- It is inconvenient or error-prone situation arising when we process the tables.
- Three types:
 1. Insertion Anomalies
 2. Update Anomalies
 3. Delete Anomalies



Insert Anomalies

- It may be impossible to store certain information without storing some other, unrelated information.
- E.g. In University Database System, it is not possible to add entry of any new department unless an employee is hired for it.

<i>ID</i>	<i>name</i>	<i>salary</i>	<i>dept_name</i>	<i>building</i>	<i>budget</i>
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000



Update Anomalies

- If one copy of such repeated data is updated, all copies need to be updated to prevent inconsistency.
- E.g. Building of Physics department is changed.
- The partial updation of records leaves database in inconsistent state.

<i>ID</i>	<i>name</i>	<i>salary</i>	<i>dept_name</i>	<i>building</i>	<i>budget</i>
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000



Delete Anomalies

- It may be impossible to delete certain information without losing some other, unrelated information.
- It occurs when data of some attributes get lost because of deletion of data of other attributes in the same records.
- E.g. Deletion of record of music department, then there will not be any record related to Packard building.

<i>ID</i>	<i>name</i>	<i>salary</i>	<i>dept_name</i>	<i>building</i>	<i>budget</i>
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000



First Normal Form (1NF)

- The relation is said to be in 1NF if and only if, It contains no repeating attributes or groups attributes.
- The relation/table is said to be in 1NF when each cell of the table/relation contains precisely one value (atomic)

A relation schema is in 1NF :

- if and only if all the attributes of the relation R are atomic in nature.
- Atomic: the smallest level to which data may be broken down and remain meaningful



Database Example

SID	CID	S_name	C_name	Grade	F_Name	F_phone
1	IS318, IS301	Anand	Database, EC	A,B	Vishal, Pradipkumar	60192, 45869
2	IS318	Vijay	Database	A	Vishal	60192
3	IS318	Shrikant	Database	B	Vishal	60192
4	IS301, IS318	Vinay	EC, Database	A,B	Pradipkumar , Vishal	45869, 60192



Database Example

■ Convert table in Normal forms

SID = Student ID, S_Name= Student Name, CID = Course ID,

C_Name = Course Name,

Grade = Student's Grade in Course Faculty = Faculty Name, F_Phone = Faculty Phone

Functional Dependencies are:

$SID \rightarrow S_name$

$SID \text{ and } CID \rightarrow Grade \quad CID \rightarrow C_name$

$CID \rightarrow F_Name$

$F_Name \rightarrow F_phone$



First Normal Form (1NF)

SID	CID	S_name	C_name	Grade	Faculty	F_phone
1	IS318	Anand	Database	A	Vishal	60192
1	IS301	Anand	EC	B	Pradipkumar	45869
2	IS318	Vijay	Database	A	Vishal	60192
3	IS318	Shrikant	Database	B	Vishal	60192
4	IS301	Vinay	EC	A	Pradipkumar	45869
4	IS318	Vinay	Database	B	Vishal	60192



Second Normal Form (2NF)

- A Relation is said to be in Second Normal Form if and only if :
 - ✓ It is in the First normal form, and
 - ✓ No partial dependency exists between non-key attributes and key attributes.
- A relation schema R is in second normal form.
 - ✓ Every non-key attribute is fully functionally dependent on all parts of the primary key.
 - ✓ Non-dependent attributes are moved into a smaller table.



Second Normal Form (2NF)

- **Full functional dependency:** a FD $Y \rightarrow Z$ where removal of any attribute from Y means the FD does not hold any more
- Examples:
 - $\{eid, PNUMBER\} \rightarrow HOURS$ is a full FD since neither $eid \rightarrow HOURS$ nor $PNUMBER \rightarrow HOURS$ hold
 - $\{eid, PNUMBER\} \rightarrow ENAME$ is not a full FD (it is called a partial dependency) since $eid \rightarrow ENAME$ also holds



Second Normal Form (2NF)

- SID and CID → Grade

SID	CID	Grade
1	IS318	A
1	IS301	B
2	IS318	A
3	IS318	B
4	IS301	A
4	IS318	B



Second Normal Form (2NF)

- $SID \rightarrow S_name$

SID	S_name
1	Anand
2	Vijay
3	Shrikant
4	Vinay

- $CID \rightarrow C_name$
- $CID \rightarrow F_Name$

CID	C_name	Faculty	F_phone
IS318	Database	Shrikant	60192
IS301	EC	Vikram	45869
IS318	Database	Shrikant	60192
IS318	Database	Shrikant	60192
IS301	EC	Vikram	45869
IS318	Database	Shrikant	60192



SCTR's

Pune Institute of Computer Technology, Pune - 411043.

Department of Electronics & Telecommunication



Advantages of Second Normal Form (2NF)

- Less data repetition.
- Improves data integrity.
- Prevents update anomalies.



Third Normal Form (3NF)

- A relation R is said to be in the Third Normal Form (3NF) if and only if
 - It is in 2NF and
 - No transitive dependency exists between non-key attributes and key attributes.



Third Normal Form (3NF)

- $SID \text{ and } CID \rightarrow \text{Grade}$

SID	CID	Grade
1	IS318	A
1	IS301	B
2	IS318	A
3	IS318	B
4	IS301	A
4	IS318	B

- $SID \rightarrow S_name$

SID	S_name
1	Anand
2	Vijay
3	Shrikant
4	Vinay



Third Normal Form (3NF)

CID	C_name	FID
IS318	Database	1
IS301	Program	2
IS318	Database	1
IS318	Database	1
IS301	Program	2
IS318	Database	1

Final table list in 3NF:

Grade (SID, CID, Grade)

Student (SID, S_name)

Course (CID, C_name, FID)

Faculty (FID, Faculty, F_phone)

FID	Faculty	F_phone
1	Vishal	60192
2	Pradipkumar	45869



BCNF

- A relation is said to be in Boyce Codd Normal Form (BCNF)
 - if and only if all the determinants are candidate keys.
 - BCNF relation is a strong 3NF, but not every 3NF relation is BCNF.



Normal Forms

Normal Form	Basic Motivation
1NF	Removing non-atomicity
2NF	Removing partial dependency (Part of key attribute → Non-key attribute)
3NF	Removing transitive dependency
BCNF	Removing any kind of redundancy



Fourth Normal Form (4NF)

- A relation is said to be in Fourth Normal Form (4NF)
 - if it is in 3NF.
 - If it does not contain multivalued dependencies.



SCTR's

Pune Institute of Computer Technology, Pune - 411043.

Department of Electronics & Telecommunication



Thank You..!!