UNIT - IV

Schema Refinement (Normalization) and Normal Forms: Problems Caused by Redundancy, Decompositions, Problems Related to Decomposition, Functional dependency, Properties of Functional dependency, Normal forms based on functional dependency - 1NF, 2NF and 3NF, concept of surrogate key, Boyce-Codd normal form(BCNF), 4NF; Properties of Decompositions - Lossless join decomposition and dependency preserving decomposition.

Normalization (Schema Refinement)

Problems Caused by Redundancy:-

- Storing the same Information Redundantly that is in more than one place within database can lead to several Problems.

1. **Redundant Storage**

   Some information is stored repeatedly.

2. **Update anomalies**

   If data items are scattered and are not linked to each other properly, then there may be instances when we try to update one data item that has copies of it scattered at several places, few instances of it get updated properly while few are left with their old values.

This leaves database in an inconsistent state.

This anomaly is caused due to data redundancy. Redundant information makes updates more difficult since, for example, changing the name of the student 501 would require that all tuples containing 501 in Regno must be updated. If for some reason, all tuples are not updated, we might have a database that gives two names for a student, which is inconsistent information. This problem is called update anomaly. An update anomaly results in **data inconsistency.**

3. **Insertion anomalies**

   We tried to insert data in a record that does not exist at all. Inability to represent certain information-The primary key of the above relation be (Regno, course code). Any new tuple to be inserted in the relation must have a value for the primary key since entity integrity constraint requires that a key may not be totally or partially NULL. However, in the given relation if one wanted to insert the code and name of a new subject in the database, it would not be possible until a student enrols in that Course. Similarly information about a new student cannot be inserted in the database until the student enrols in a course. These problems are called insertion anomalies.

4. **Deletion anomalies**

   We tried to delete a record, but parts of it left undeleted because of unawareness, the data is also saved somewhere else. Loss of Useful Information: In some instances, useful information may be lost when a tuple is deleted. For example, if we delete the tuple corresponding to student 502 enrolled for CS-104, we will lose relevant information about the course i.e., course name. This is called deletion anomaly.

Example is:

Regno	NAME	Address	Course code	CNAME	Marks
501	Lakshmi	GNT	CS101	Computer organization	75
501	Lakshmi	GNT	CS102	DBMS	85
502	Kiran	VJA	CS104	Microprocessor	99

Decomposition:

- Decomposition is the process of breaking down the a relation R into two or more relation schemas that each contain a subset of the attributes of R and together include all attributes in R.

- Decomposition helps in eliminating some of the problems of bad design such as redundancy, inconsistency and anomalies.

There are two types of decomposition

1. Lossless join decomposition
2. Lossy Decomposition

- The decomposition of R into R1 and R2 is lossy when the join of R1 and R2 does not yield the same relation as in R.
- One of the disadvantage of decomposition into two or more tables is that some information is lost during retrieval of original relation.
- e.g. if we have a table STUDENT (Rollno,sname,dept). If we decompose the table into two tables one with attributes Student_info(Rollno,sname) and another as Student_dept(sname,dept). When we join the two tables then we may get some spurious or extra tupples which makes the data inconsistent.

Lossless Join Decomposition (Non-additive Join)

- The decomposition of R into R1 and R2 is lossless when the join of R1 and R2 yield the same relation as in R.
- e.g. if we have a table STUDENT (Rollno,sname,dept). If we decompose the table into two tables one with attributes Student_info(Rollno,sname) and another as Student_dept(Rollno,dept).
- When we join the two tables then we get the same relation as that of student.
- Here no spurious or extra tuples are generated.
- Hence, care must be taken before decomposing a relation into parts.

Functional dependency

Definition

- A functional dependency is a constraint between two sets of attributes from the database
- A **functional dependency,** denoted by $X \rightarrow Y$, between two sets of attributes X and Y that are subsets of R specifies a *constraint* on the possible tuples that can form a relation state r of R.
- The constraint is that, for any two tuples t1 and t2 if, t1 [X] =t2 [X], then we must also have t1 [Y] = t2[Y].
- This means that the values of the Y component of a tuple depend on, or are *determined by,* the values of the X component; or alternatively, the values of the X component of a tuple uniquely (or **functionally**) **determine** the values of the Y component.
- We also say that there is a functional dependency from X to Y or that Y is **functionally dependent** on X.
- The abbreviation for functional dependency is **FD** or **f.d.** The set of attributes X is called the **left-hand side of** the FD, and Y is called the **right-hand side.**
- Functional dependency is represented by arrow sign (→) that is A→B, where A functionally determines B.

Determinant: Attribute or set of attributes on the **left hand side** of the arrow.

Dependent: Attribute or set of attributes on the **right hand side** of the arrow.

Properties of FD: [Armstrong's Axioms]

- If F is set of functional dependencies then the closure of F, denoted as F+, is the set of all functional dependencies logically implied by F. Armstrong's Axioms are set of rules, when applied repeatedly generates closure of functional dependencies.

1. **Reflexive rule:** If A is a set of attributes and B is a subset of A, then A holds B.

If A ⊃ B then A →B

2. **Augmentation rule:** if a→b holds and y is attribute set, then ***ay→by*** also holds. That is adding attributes in dependencies, does not change the basic dependencies.

3. **Transitivity rule:** Same as transitive rule in algebra, if a→b holds and b→c holds then a→c also hold a→b is called functionally determines b

4. **Decomposition rule:** *If X →YZ then X→Y and X →Z*

5. **Union rule:** *If X → Y and X → Z then X → YZ*

6. **Pseudo transitive rule:** *If X →Y, WY →Z then WX → Z*

Trivial Functional Dependency

- ✓ **Trivial:** If an FD X->Y holds where Y subset of X , then its called TRIVIAL FD
- ✓ **Non-trivial:** If an FD X - >Y holds where Y not subset of X , then its called NON-TRIVIAL FD

Normalization

- ✓ If a database design is not perfect it may contain anomalies, which leads to inconsistence of database itself. Managing a database with anomalies is next to impossible.
- ✓ Normalization is the process of efficiently organizing data in the DB.

There are two goals of the normalization process:

- ✓ Eliminate redundant data (for example, storing the same data in more than one table) and
- ✓ Ensure data dependencies make sense (only storing related data in a table).

Both of these are worthy goals as they reduce the amount of space a database consumes and ensure that data is logically stored.

Normal Forms Based on FDS:

- The normalization process, as first proposed by Codd (1972a), takes a relation schema through a series of tests to "certify" whether it satisfies a certain **normal form.**
- The normal forms based on FDs are 1st normal form (1NF), second normal form (2NF), third normal form (3NF), and Boyce-Codd normal form (BCNF).
- **Normalization of data** can hence be looked upon as a process of analyzing the given relation schemas based on their FDs and primary keys to achieve the desirable properties of

1) minimizing redundancy and
2) minimizing the insertion, deletion, and update anomalies

ADVANTAGES OF NORMALIZATION

The following are the advantages of the normalization.

- ☑ More efficient data structure.
- ☑ Avoid redundant fields or columns.
- ☑ More flexible data structure i.e. we should be able to add new rows and data values easily
- ☑ Better understanding of data.
- ☑ Ensures that distinct tables exist when necessary.

- ☑ Easier to maintain data structure i.e. it is easy to perform operations and complex queries can be easily handled.
- ☑ Minimizes data duplication.
- ☑ Close modeling of real world entities, processes and their relationships.

DISADVANTAGES OF NORMALIZATION

The following are disadvantages of normalization.

- ☑ You cannot start building the database before you know what the user needs.
- ☑ On Normalizing the relations to higher normal forms i.e. 4NF, 5NF the performance degrades.
- ☑ It is very time consuming and difficult process in normalizing relations of higher degree.
- ☑ Careless decomposition may leads to bad design of database which may leads to serious problems.

EXAMPLE DATA:

A company obtains parts from a number of suppliers. Each supplier is located in one city. A city can have more than one supplier located there and each city has a status code associated with it. Each supplier may provide many parts. The company creates a simple relational table to store this information that can be expressed in relational notation as:

FIRST (s#, status, city, p#, qty)

where

s# supplier identifcation number (this is the primary key)
status status code assigned to city
city name of city where supplier is located
p# part number of part supplied
qty> quantity of parts supplied to date

First Normal Form [1NF]:-

Definition: **A relation is said to be in 1NF if it contains no non-atomic values and each row can provide a unique combination of values**.

- For example all the fields in the below table are atomic with single values and each row contains unique combination of values so it in 1NF
- 1NF does not allow multivalued attributes. If multivalued attributes present then the relation need to be decomposed.

FIRST

s#	status	city	p#	qty
s1	20	London	p1	300
s1	20	London	p2	200
s1	20	London	p3	400
s1	20	London	p4	200
s1	20	London	p5	100
s1	20	London	p6	100
s2	10	Paris	p1	300
s2	10	Paris	p2	400
s3	10	Paris	p2	200
s4	20	London	p2	200
s4	20	London	p4	300
s4	20	London	p5	400

- Although the table FIRST is in 1NF it contains redundant data. For example, information about the supplier's location and the location's status have to be repeated for every part supplied.
- Redundancy causes what are called *update anomalies*.

Update anomalies are problems that arise when information is inserted, deleted, or updated.

- INSERT. The fact that a certain supplier (s5) is located in a particular city (Athens) cannot be added until they supplied a part.
- DELETE. If a row is deleted, then not only is the information about quantity and part lost but also information about the supplier.
- UPDATE. If supplier s1 moved from London to New York, then six rows would have to be updated with this new information.

Second Normal Form (2NF):-

Definition: **A relation is said to be in 2NF, if it is already in 1NF and each and every attribute fully depends on the primary key of the relation**.

- Speaking inversely, if a table has some attributes which is not dependent on the primary key of that table, then it is not in 2NF.
- That is, every non-key column must be dependent upon the entire primary key. FIRST is in 1NF but not in 2NF because status and city are functionally dependent upon only on the column **s#** of the composite primary key (s#, p#).
- This can be illustrated by listing the functional dependencies in the table:

s# →city, status
city →status
(s#,p#)→qty

The process for transforming a 1NF table to 2NF is:

1. Identify any determinants other than the composite key, and the columns they determine.
2. Create and name a new table for each determinant and the unique columns it determines.
3. Move the determined columns from the original table to the new table. The determinate becomes the primary key of the new table.
4. Delete the columns you just moved from the original table except for the determinate which will serve as a foreign key.
5. The original table may be renamed to maintain semantic meaning.

To transform FIRST into 2NF we move the columns s#, status, and city to a new table called SECOND. The column s# becomes the primary key of this new table

SECOND

s#	status	city
s1	20	London
s2	10	Paris
s3	10	Paris
s4	20	London
s5	30	Athens

PARTS

s#	p#	qty
s1	p1	300
s1	p2	200
s1	p3	400
s1	p4	200
s1	p5	100
s1	p6	100
s2	p1	300
s2	p2	400
s3	p2	200
s4	p2	200
s4	p4	300
s4	p5	400

- Tables in 2NF but not in 3NF still contain modification anomalies. In the example of SECOND, they are:

- INSERT. The fact that a particular city has a certain status (Rome has a status of 50) cannot be inserted until there is a supplier in the city.

- DELETE. Deleting any row in SUPPLIER destroys the status information about the city as well as the association between supplier and city.

Third Normal Form (3NF):-

Definition: **A relation is said to be in 3NF, if it is already in 2NF and there exists no** transitive dependency **in that relation**.

- Speaking inversely, if a table contains transitive dependency, then it is not in 3NF, and the table must be split to bring it into 3NF

What is a transitive dependency? Within a relation if we see
A → B [B depends on A]
And
B → C [C depends on B]
Then we may derive
A → C [C depends on A]

Table PARTS is already in 3NF. The non-key column, qty, is fully dependent upon the primary key (s#, p#). SUPPLIER is in 2NF but not in 3NF because it contains a transitive dependency. A transitive dependency is occurs when a non-key column that is a determinant of the primary key is the determinate of other columns. The concept of a transitive dependency can be illustrated by showing the functional dependencies in SUPPLIER:

s# —> city
city —>status

Then,

s# —>status

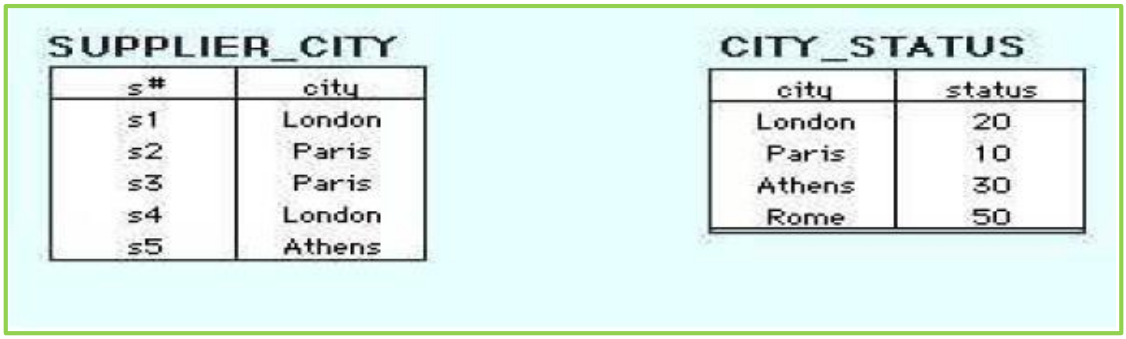
Note that SUPPLIER.status is determined both by the primary key s# and the non-key column city. The process of transforming a table into 3NF is:

1. Identify any determinants, other the primary key, and the columns they determine.
2. Create and name a new table for each determinant and the unique columns it determines.

3. Move the determined columns from the original table to the new table. The determinate becomes the primary key of the new table.
4. Delete the columns you just moved from the original table except for the determinate which will serve as a foreign key.
5. The original table may be renamed to maintain semantic meaning.

To transform SUPPLIER into 3NF, we create a new table called CITY_STATUS and move the columns city and status into it. Status is deleted from the original table, city is left behind to serve as a foreign key to CITY_STATUS, and the original table is renamed to SUPPLIER_CITY to reflect its semantic meaning. The results are shown in Figure 3 below.

Figure 3: Tables in 3NF



SUPPLIER_CITY

s#	city
s1	London
s2	Paris
s3	Paris
s4	London
s5	Athens

CITY_STATUS

city	status
London	20
Paris	10
Athens	30
Rome	50

The results of putting the original table into 3NF has created three tables. These can be represented in "psuedo-SQL" as:

PARTS (#s, p#, qty)
Primary Key (s#,#p)
Foreign Key (s#) references SUPPLIER_CITY.s#

SUPPLIER_CITY(s#, city)
Primary Key (s#)
Foreign Key (city) references CITY_STATUS.city

CITY_STATUS (city, status)
Primary Key (city)

Advantages of Third Normal Form

- The advantages to having relational tables in 3NF is that it eliminates redundant data which in turn saves space and reduces manipulation anomalies.

For example, the improvements to our sample database are:

- INSERT. Facts about the status of a city, Rome has a status of 50, can be added even though there is not supplier in that city. Likewise, facts about new suppliers can be added even though they have not yet supplied parts.
- DELETE. Information about parts supplied can be deleted without destroying information about a supplier or a city.
- UPDATE. Changing the location of a supplier or the status of a city requires modifying only one row.

Boyce-Codd normal form (BCNF)

Definition: - **A relationship is said to be in BCNF if it is already in 3NF and the left hand side of every Dependency (determinant) is a candidate key**.

- A relation which is in 3NF is almost always in BCNF. These could be some situation when a 3NF relation may not be in BCNF the following conditions are found true.

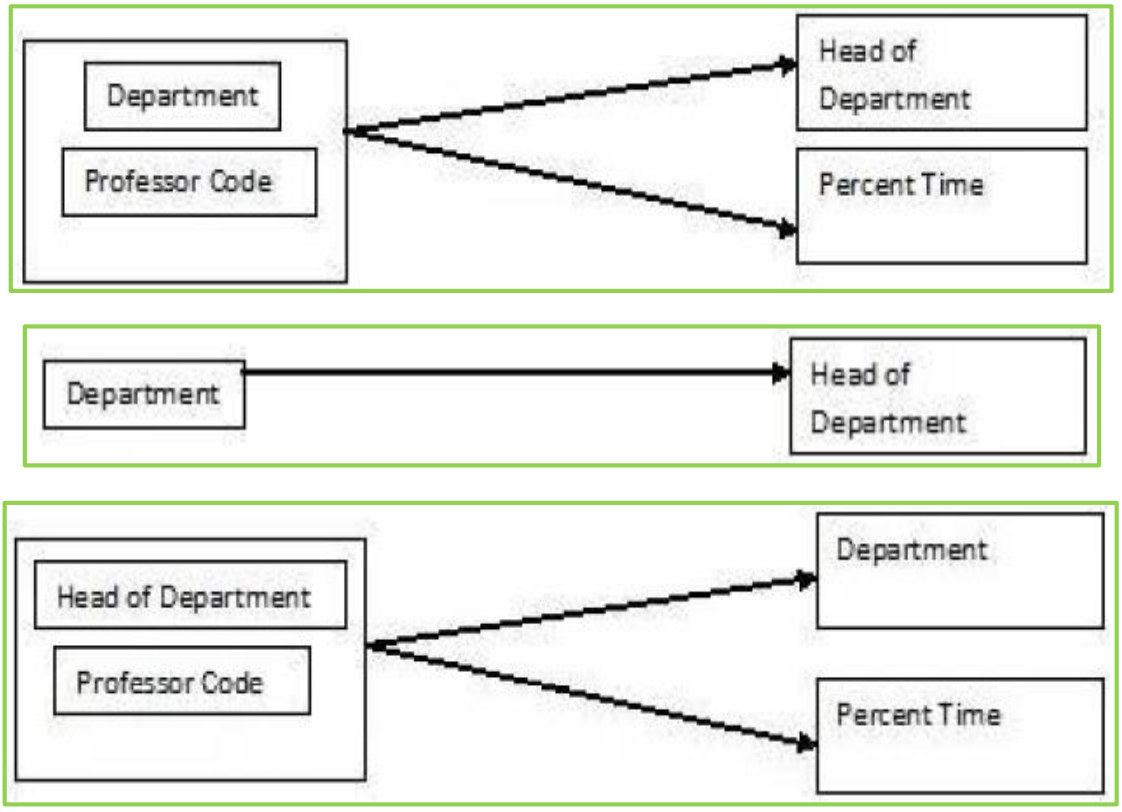
1. The candidate keys are composite.
2. There are more than one candidate keys in the relation.
3. There are some common attributes in the relation.

Professor Code	Department	Head of Dept.	Percent Time
P1	Physics	Ghosh	50
P1	Mathematics	Krishnan	50
P2	Chemistry	Rao	25
P2	Physics	Ghosh	75
P3	Mathematics	Krishnan	100

Consider, as an example, the above relation. It is assumed that:

1. A professor can work in more than one department.
2. The percentage of the time he spends in each department is given.
3. Each department has only one Head of Department.

The relation diagram for the above relation is given as the following:



- The given relation is in 3NF. Observe, however, that the names of Dept. and Head of Dept. are duplicated. Further, if Professor P2 resigns, rows 3 and 4 are deleted. We lose the information that Rao is the Head of Department of Chemistry.

- The normalization of the relation is done by creating a new relation for Dept. and Head of Dept. and deleting Head of Dept. form the given relation. The normalized relations are shown in the following.

Professor Code	Department	Percent Time
P1	Physics	50
P1	Mathematics	50
P2	Chemistry	25
P2	Physics	75
P3	Mathematics	100

Department	Head of Dept.
Physics	Ghosh
Mathematics	Krishnan
Chemistry	Rao

Fourth Normal Form (4NF):

- In general, **a multivalued dependency** occurs when a relation R has attributes A, B, and C such that A determines a set of values for B, A determines a set of values for C, and B and C are independent of each other.

Definition: - **A relation is said to be in 4NF if it is in BCNF/3NF and contains no multi valued attributes**.

(Or)

Let R be a relation schema, X and Y be nonempty subsets of the attributes of R, and F be a set of dependencies that includes both FDs and MVDs. R is in **4NF** if, for every non trivial MVD X->->Y that holds over R, X is a super key.

- Trivial Multi value dependency :it is one MVD that satisfied in $Y \subseteq X$ or XY = R, or (trivial) N

Now consider another table example involving Course, Student_name and text_book.

Ename	Projname	Children name
Smith	X	John
Smith	Y	Anne
Smith	X	Anne
Smith	Y	John

The attributes Projname' and Children name are multivalued facts about the attribute 'Ename'.

However, since a Project has no influence over the Chidren name, these multi-valued facts about 10 courses are independent of each other. Thus the table contains an MVD. Multi-value facts are represented by.

Here, in above database following MVDs exists:

Ename --> --> Projname

Ename--> --> Children name

Here, Project and children are independent of each other. So all the anomalies occur in the above table

Solution of above anomalies with Fourth Normal Form

This problem of MVD is handled in Fourth Normal Form. To put it into 4NF, two separate tables are formed as shown below:

Emp-Proj(ename,projname)

Emp-Children (ename,childrenname)

ename	Children name
Smith	John
Smith	Anne

ename	Projname
Smith	X
Smith	Y

5th Normal Form (5NF):

Definition: - A relation is said to be in 5NF if and only if every join dependency in relation is implied by the candidate keys of relation.

Or

Simply we can say "A table is in fifth normal form (5NF) if it is in 4NF and it cannot have a lossless decomposition into any number of smaller tables.

The functional dependencies are

PNO ->> SNO

SNO->>JNO

PNO->>JNO

where PNO is Project number, SNO is Supplier number and JNO is Parts number.

The above table is not in 5nf. To make it into 5nf the table is decomposed into 3 TABLES.ie. Supplier-Project, Supplier-Parts and Project- Parts

Supplier-Project	
SNO	PNO
S1	P1
S1	P2
S2	P1

Project-Parts	
PNO	JNO
P1	J1
P1	J2
P2	J1

Supplier-Parts	
SNO	JNO
S1	J1
S1	J2
S2	J1

When we join all these tables, we get the original relation.

SUMMARY

TABLE 10.1 SUMMARY OF NORMAL FORMS BASED ON PRIMARY KEYS AND CORRESPONDING NORMALIZATION

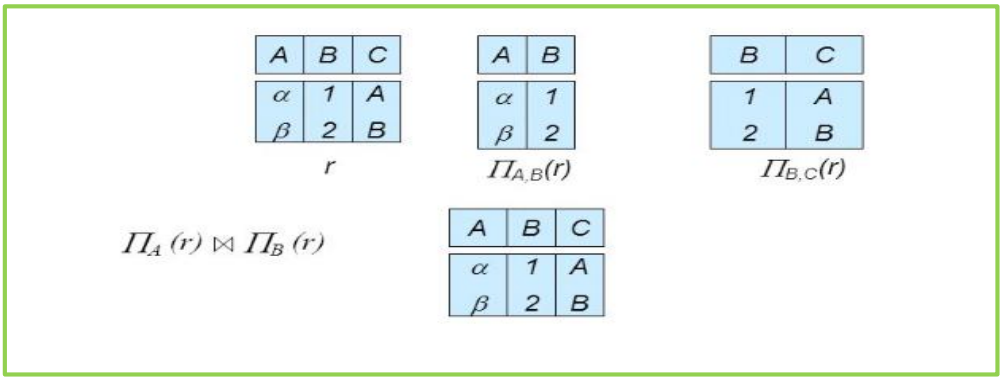
NORMAL FORM	TEST	REMEDY (NORMALIZATION)
First (1NF)	Relation should have no nonatomic attributes or nested relations.	Form new relations for each nonatomic attribute or nested relation.
Second (2NF)	For relations where primary key contains multiple attributes, no nonkey attribute should be functionally dependent on a part of the primary key.	Decompose and set up a new relation for each partial key with its dependent attribute(s). Make sure to keep a relation with the original primary key and any attributes that are fully functionally dependent on it.
Third (3NF)	Relation should not have a nonkey attribute functionally determined by another nonkey attribute (or by a set of nonkey attributes.) That is, there should be no transitive dependency of a nonkey attribute on the primary key.	Decompose and set up a relation that includes the nonkey attribute(s) that functionally determine(s) other nonkey attribute(s).

Desirable Properties of Decomposition

Two important properties

Lossless-join decomposition

- Required for semantic correctness
- When we join the decomposed relations, we should get back exact original relation contents
- No spurious tuples



Dependency preservation:

- It is a property of decomposition. The FD s that hold on the relation R should be preserved directly or indirectly even after decomposition.
- i.e. we should decompose the relation R in such a way that the FDs that hold good on R must be derived or holds good on the decomposed relations.

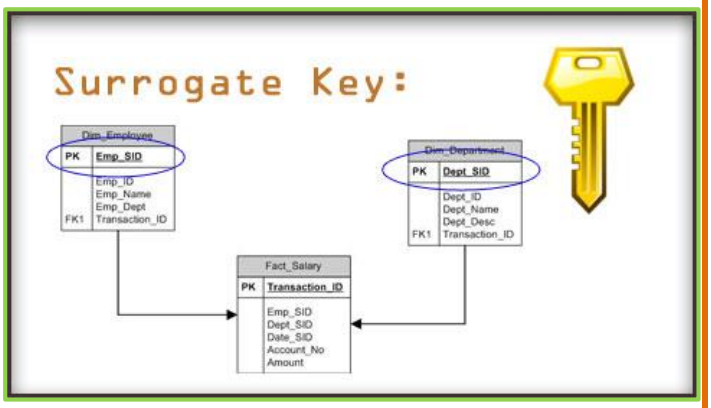
Candidate Keys

- For some entities, there may be more than one field or collection of fields that can serve as the primary key. These are called candidate keys.

Surrogate Keys

Definition: A "surrogate key" is a field (usually numeric) added to a table as the primary key when no natural keys are available.

- Occasionally, you will come across a table that has **NO candidate keys**. In this case, you must resort to a surrogate key
- When you have to resort to a surrogate key, don't try to put any meaning into it. The values in surrogate key fields frequently become important identification numbers: Social Security Numbers, Student IDs, VINs, SKUs (in stores), etc. They are frequently associated with some sort of ID card or tag.



Surrogate Key Example

Suppose you run a chicken farm with lots of chickens. You want to be able to track the chickens—when they were hatched, weights at certain ages, how many eggs they lay, etc. The chickens don't have names, social security numbers, uniqnames, or any other simple way to tell them apart. Therefore, you put a numbered tag on each chicken. This number becomes the surrogate key in your Chickens table.

Natural vs. Surrogate Keys

Using a surrogate key when a natural key is available can defeat the whole purpose of having a primary key. This example shows how using an AutoNumber primary key allows duplicate entries to be made (StaffID is the primary key): If SocialSecurityNumber were the primary key, this duplicate entry would not be allowed by the DBMS.