Database Management Systems(BCS403)

MODULE 3

Module – 3 Introduction to SQL

Contents:

Database Design Theory

- Introduction.
- Informal design guidelines for relation schema.
- Functional Dependencies.

Normalization

- Introduction to Normalization.
- Normal Forms based on Primary Keys.
- Second and Third Normal Forms.
- Boyce-Codd Normal Form.
- Multivalued Dependency and Fourth Normal Form.
- Join Dependencies and Fifth Normal Form.
- Examples on normal forms.

Module – 3 Introduction to SQL

Contents:

Introduction to SQL

- SQL data definition and data types
- Specifying constraints in SQL
- Retrieval queries in SQL.
- INSERT, DELETE, and UPDATE statements in SQL.
- Schema change statements in SQL.
- Additional features of SQL.

Module – 3 Database Design Theory

Introduction

There are two approaches for database design

Bottom-up approach:

- A bottom-up design methodology (also called **design by synthesis**) considers the basic relationships among individual attributes as the starting point and uses those to construct relation schemas.
- This approach is not very popular in practice because it suffers from the problem of having to collect a large number of binary relationships among attributes as the starting point. For practical situations, it is next to impossible to capture binary relationships among all such pairs of attributes.

Top-down approach:

- Atop-down design methodology (also called **design by analysis**) starts with a number of groupings of attributes into relations that exist together naturally.

Module – 3 Database Design Theory

Introduction

- The relations are then analyzed individually and collectively, leading to further decomposition until all desirable properties are met.

Two implicit goals of the database design activity are

1. Information preservation.
2. Minimum redundancy.

Information preservation:

- Information preservation in terms of maintaining all concepts, including attribute types, entity types, and relationship types as well as generalization/specialization relationships, which are described using a model such as the EER(Enhanced-ER) model.
- The relational design must preserve all of these concepts, which are originally captured in the conceptual design after the conceptual to logical design mapping.

Module – 3 Database Design Theory

Introduction

Minimum redundancy.

- Minimizing redundancy implies minimizing redundant storage of the same information and reducing the need for multiple updates to maintain consistency across multiple copies of the same information in response to real-world events that require making an update.

Specialization

- Specialization is the process of defining a set of subclasses of an entity type; this entity type is called the **superclass** of the specialization.
- The set of subclasses that forms a specialization is defined on the basis of some distinguishing characteristic of the entities in the superclass.

Module – 3 Database Design Theory

Introduction

For example, the set of subclasses {SECRETARY, ENGINEER, TECHNICIAN} is a specialization of the superclass **EMPLOYEE** that distinguishes among employee entities based on the **job type** of each employee.

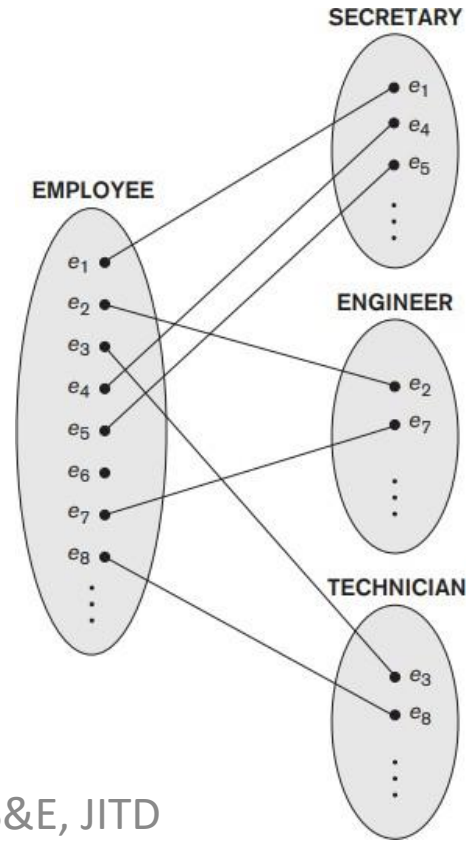


Figure: 1 Instances of a specialization.

Module – 3 Database Design Theory

Introduction

Generalization

We can think generalization as a reverse process of abstraction in which we suppress the differences among several entity types, identify their common features, and generalize them into a single superclass of which the original entity types are special subclasses.

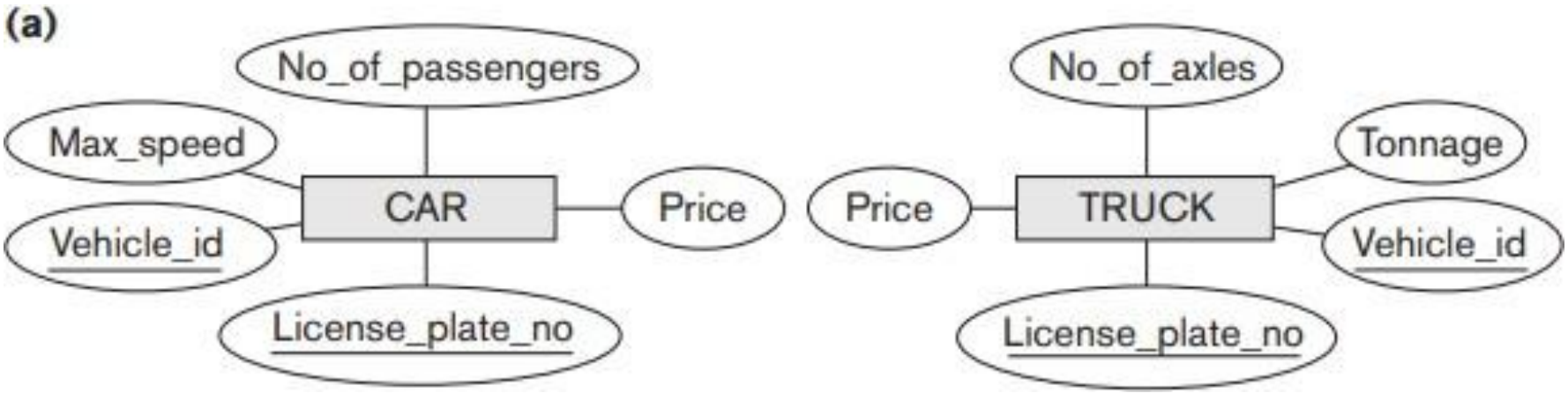


Figure: 2 (a) Two entity types, CAR and TRUCK

Module – 3 Database Design Theory

Introduction

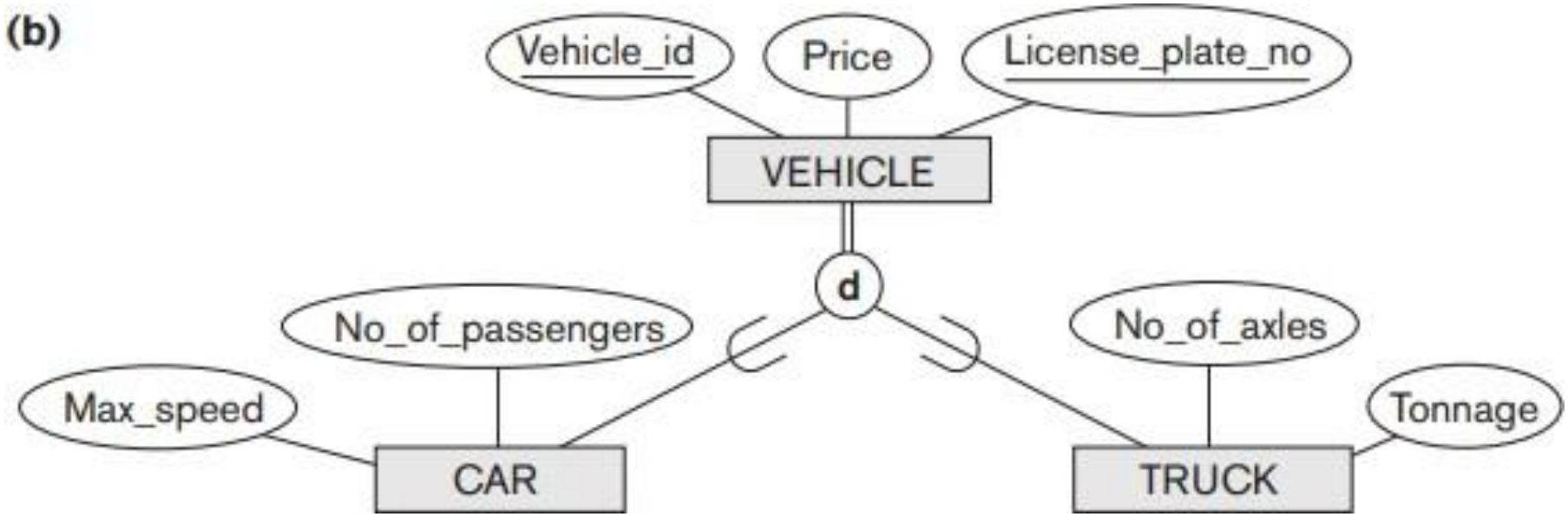


Figure: 3 (b) Generalizing CAR and TRUCK into the superclass VEHICLE.

Note: Functional dependency and a formal constraint among attributes that is the **main tool for formally measuring the appropriateness** of attribute groupings into relation schemas.

Module – 3 Database Design Theory

Informal Design Guidelines for Relation Schemas

There are **four informal guidelines** that may be used as measures to determine the quality of relation schema design:

1. Making sure that the **semantics of the attributes** is clear in the schema.
2. Reducing the **redundant information** in tuples.
3. Reducing the **NULL values** in tuples.
4. Disallowing the possibility of generating **spurious tuples**.

Imparting Clear Semantics to Attributes in Relations

- Attributes belonging to one relation should have certain real-world meaning and a proper interpretation associated with them.
- The **semantics of a relation** refers to its meaning, which is resulting from the interpretation of attribute values in a tuple.
- Consider a **simplified COMPANY relational** database schema…..

Module – 4 Database Design Theory

Informal Design Guidelines for Relation Schemas

The **meaning of the EMPLOYEE relation** schema is simple:

- Each tuple represents an employee, with values for the employee‟s name (Ename), Social Security number (Ssn), birth date (Bdate), and address (Address), and the number of the department that the employee works for (Dnumber).
- The Dnumber attribute is a foreign key that represents an implicit relationship between EMPLOYEE and DEPARTMENT.

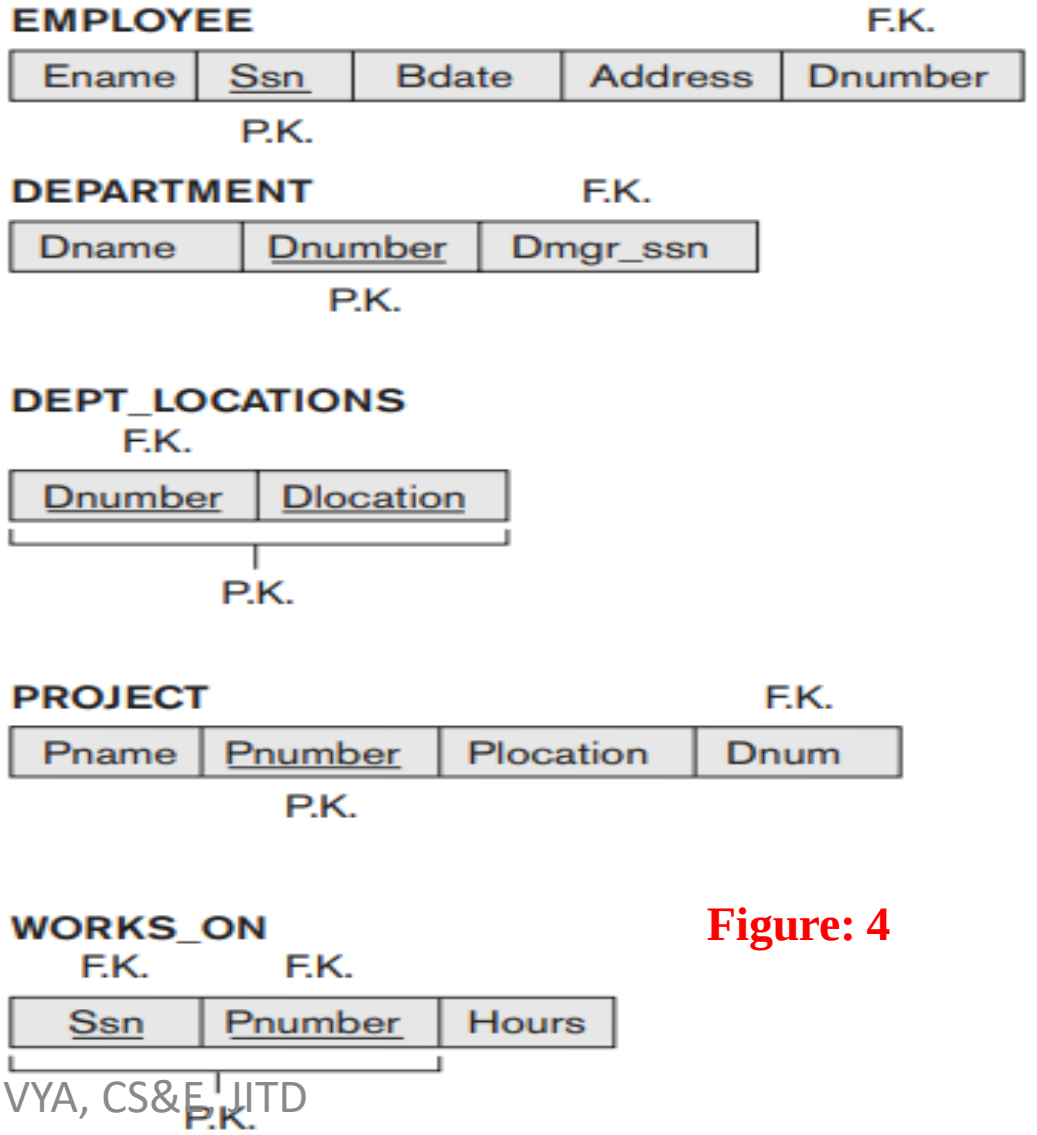


Figure: 4

Module – 3 Database Design Theory

Informal Design Guidelines for Relation Schemas

- From the above example COMPANY database, we found that it is very easy to explain the meaning of each attribute in each relation.
- Thus the following informal guideline can be formulated..

Guideline 1:

- Design a relation schema so that it is easy to explain its meaning.
- Do not combine attributes from multiple entity types and relationship types into a single relation.
- If a relation schema corresponds to one entity type or one relationship type, it is straightforward to explain its meaning.
- Otherwise, if the relation corresponds to a mixture of multiple entities types and relationships, semantic ambiguities will result and the relation cannot be easily explained.

Module – 3 Database Design Theory

Informal Design Guidelines for Relation Schemas

Consider the following EMP_DEPT schema which has **sematic ambiguity** since it **combines attributes from more than one entity** types.

EMP_DEPT

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn

- A tuple in the EMP_DEPT relation schema shown above represents a single employee but includes, along with the Dnumber (the identifier for the department he/she works for), additional information—namely, the name (Dname) of the department for which the employee works and the Social Security number (Dmgr_ssn) of the department manager.
- This **violate Guideline 1** by mixing attributes from distinct real-world entities: EMP_DEPT mixes attributes of employees and departments.

Module – 3 Database Design Theory

Informal Design Guidelines for Relation Schemas

- Similarly the following EMP_PROJ schema which has **sematic ambiguity** since it **combines attributes from more than one entity** types.

EMP_PROJ

Ssn	Pnumber	Hours	Ename	Pname	Plocation

- This **violate Guideline 1** by mixing attributes different relations.
- EMP_PROJ mixes attributes of employees and projects and the WORKS_ON relationship.
- Hence, they fare poorly against the above measure of design quality.

Module – 3 Database Design Theory

Informal Design Guidelines for Relation Schemas

Redundant Information in Tuples and Update Anomalies

- One goal of schema design is to minimize the storage space used by the base relations(tables stored as files in storage media).
- Consider the following state of EMP_DEPT relation which has **redundant data** since it **combines attributes**(natural join of base relations**) from more than one entity** types.

Redundancy

EMP_DEPT

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 FireOak, Humble, TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	888665555

Module – 3 Database Design Theory

Informal Design Guidelines for Relation Schemas

Redundancy Redundancy

EMP_PROJ

Ssn	Pnumber	Hours	Ename	Pname	Plocation
123456789	1	32.5	Smith, John B.	ProductX	Bellaire
123456789	2	7.5	Smith, John B.	ProductY	Sugarland
666884444	3	40.0	Narayan, Ramesh K.	ProductZ	Houston
453453453	1	20.0	English, Joyce A.	ProductX	Bellaire
453453453	2	20.0	English, Joyce A.	ProductY	Sugarland
333445555	2	10.0	Wong, Franklin T.	ProductY	Sugarland
333445555	3	10.0	Wong, Franklin T.	ProductZ	Houston
333445555	10	10.0	Wong, Franklin T.	Computerization	Stafford
333445555	20	10.0	Wong, Franklin T.	Reorganization	Houston
999887777	30	30.0	Zelaya, Alicia J.	Newbenefits	Stafford
999887777	10	10.0	Zelaya, Alicia J.	Computerization	Stafford
987987987	10	35.0	Jabbar, Ahmad V.	Computerization	Stafford
987987987	30	5.0	Jabbar, Ahmad V.	Newbenefits	Stafford
987654321	30	20.0	Wallace, Jennifer S.	Newbenefits	Stafford
987654321	20	15.0	Wallace, Jennifer S.	Reorganization	Houston
888665555	20	Null	Borg, James E.	Reorganization	Houston

Module – 3 Database Design Theory

Informal Design Guidelines for Relation Schemas

Storing natural joins of base relations **leads to an additional problem referred to as update anomalies**.

Update anomalies can be classified into

1. Insertion anomalies.
2. Deletion anomalies.
3. Modification anomalies.

Insertion anomalies

By considering EMP_DEPT schema insertion anomalies occurs in two cases;

Case – 1: It is difficult to insert a new department that has no employees as yet in the EMP_DEPT relation. The only way to do this is to place NULL values in the attributes for employee. This violates the entity integrity for EMP_DEPT because its primary key Ssn cannot be null.

Module – 3 Database Design Theory

Informal Design Guidelines for Relation Schemas

Case – 2: Consistency problem. For example, to insert a new tuple for an employee who works in department number 5, we must enter all the attribute values of department 5 correctly so that they are consistent with the corresponding values for department 5 in other tuples in EMP_DEPT.

Deletion Anomalies.

- If we delete an employee (he/she is the only one employee working for particular department) from EMP_DEPT, then we loose the information concerning that department from the database.
- This problem does not occur in the database of **Figure 4** because DEPARTMENT tuples are stored separately.

Module – 3 Database Design Theory

Informal Design Guidelines for Relation Schemas

Modification Anomalies:

- In EMP_DEPT, if we change the value of one of the attributes of a particular department say, the manager of department 5,we must update the tuples of all employees who work in that department; otherwise, the database will become inconsistent.
- If we fail to update some tuples, the same department will be shown different managers for different employee in the same department, which would be wrong.
- The three anomalies mentioned above are undesirable and cause difficulties to maintain consistency of data as well as require unnecessary updates.
- Hence the following guideline can be formulated…..

Module – 3 Database Design Theory

Informal Design Guidelines for Relation Schemas

Guideline 2:

- Design the base relation schemas so that no insertion, deletion, or modification anomalies are present in the relations.
- If any anomalies are present, note them clearly and make sure that the programs that update the database will operate correctly.

NULL Values in Tuples

- In some schema designs we may group many attributes together into a “fat” relation.
- If many of the cases some attributes do not apply to all tuples in the relation, we end up with many NULLs in those tuples.
- This can waste space at the storage level and may also lead to problems with understanding the meaning of the attributes and with specifying JOIN operations at the logical level.

Module – 3 Database Design Theory

Informal Design Guidelines for Relation Schemas

- Another problem with NULLs is how to account for them when aggregate operations such as COUNT or SUM are applied.
- SELECT and JOIN operations involve comparisons; if NULL values are present, the results may become unpredictable.
- Hence the following guideline can be formulated…..

Guideline 3:

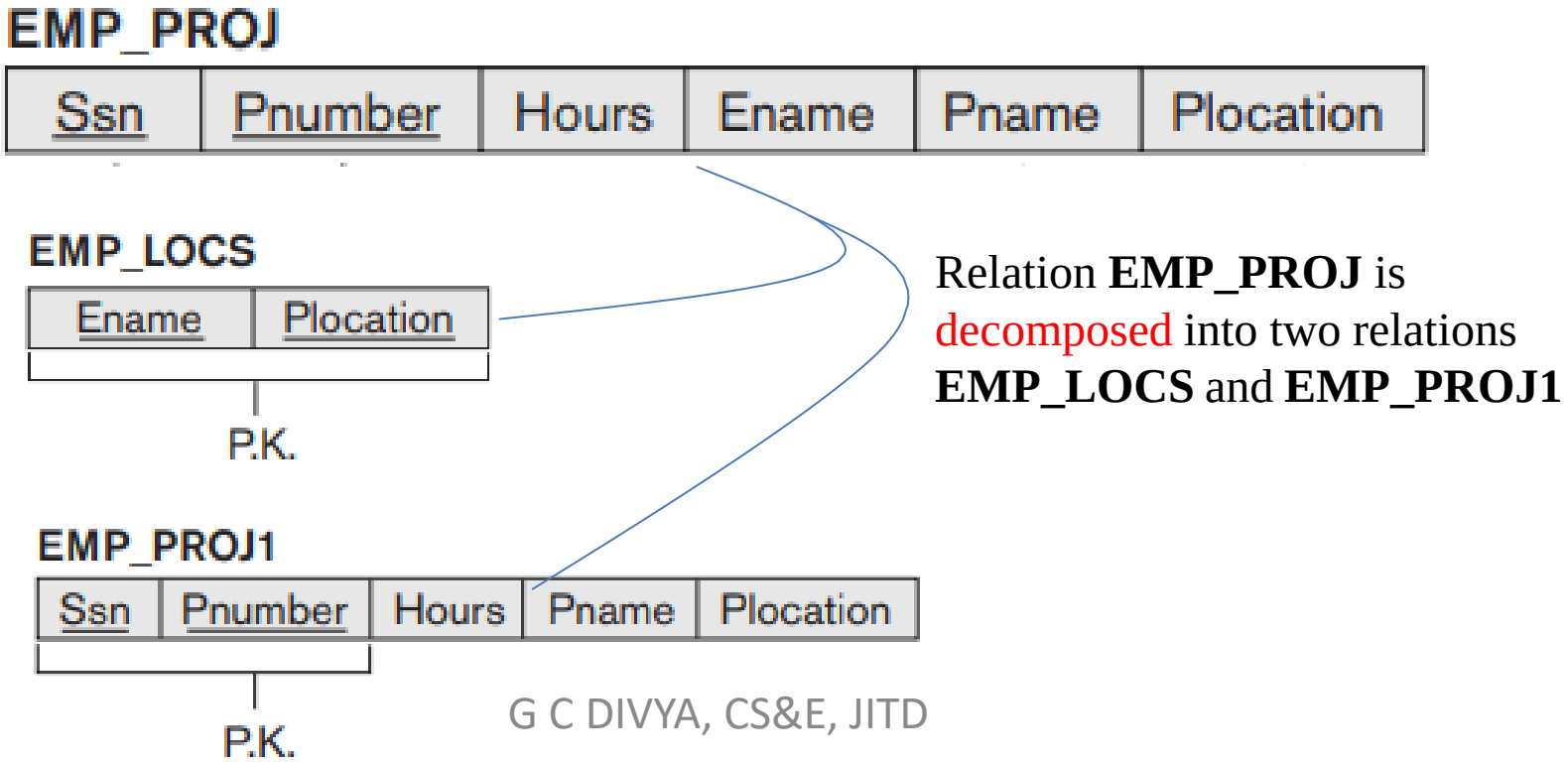
- As far as possible, avoid placing attributes in a base relation whose values may frequently be NULL.
- If NULLs are unavoidable, make sure that they apply in exceptional cases only and do not apply to a majority of tuples in the relation.

Module – 3 Database Design Theory

Informal Design Guidelines for Relation Schemas

Generation of Spurious Tuples

- Spurious Tuples are those rows in a table, which occur as a result of joining two tables in the wrong manner.
- They are extra tuples (rows) that might not be required.



Module – 3 Database Design Theory

Informal Design Guidelines for Relation Schemas

- Suppose that we used EMP_PROJ1 and EMP_LOCS as the base relations instead of EMP_PROJ and attempt a NATURAL JOIN operation on EMP_PROJ1 and EMP_LOCS, the **result produces many more tuples than the original set of tuples** in EMP_PROJ.
- Additional tuples that were not in EMP_PROJ are called spurious tuples.
- Decomposing EMP_PROJ into EMP_LOCS and EMP_PROJ1 is undesirable because when we JOIN them back using NATURAL JOIN, we do not get the correct original information.
- This is because in this case **Plocation** is the attribute that relates EMP_LOCS and EMP_PROJ1, and Plocation is neither a primary key nor a foreign key in either EMP_LOCS or EMP_PROJ1.
- Hence the following guideline can be formulated…..

Module – 3 Database Design Theory

Informal Design Guidelines for Relation Schemas

Guideline 4:

- Design relation schemas so that they can be joined with equality conditions on attributes that are appropriately related (primary key, foreign key) pairs in a way that guarantees that no spurious tuples are generated.
- Avoid relations that contain matching attributes that are not (foreign key, primary key) combinations because joining on such attributes may produce spurious tuples.

Module – 3 Database Design Theory

Normal Forms Based on Primary Keys

- Each relation is associated with set of functional dependencies and has designated primary key.
- This information is used to test for normal forms which drives normalization process f or relational schema design.

Most practical relational design projects take one of the following two approaches:

1. Perform a conceptual schema design using a conceptual model such as ER or EER and map the conceptual design into a set of relations.
2. Design the relations based on external knowledge derived from an existing implementation of files or forms or reports.

Once the relations are designed using any one of the above approaches, evaluate the relations for goodness and decompose them further as needed to achieve higher normal forms using the normalization process.

Module – 3 Database Design Theory

Normalization of Relations

- The **normalization pr**ocess was first proposed by Codd (1972).
- **Normalization** process takes a relation schema through a series of tests to certify whether it satisfies a certain normal form.
- The process, which proceeds in a **top-down fashion** by evaluating each relation against the criteria for normal forms and decomposing relations as necessary.
- Thus this process can be considered as **relational design by analysis**.

Module – 3 Database Design Theory

Normalization of data

- This can be considered a process of analyzing the given relation schemas **based on their FDs and primary keys** to achieve the following goals;
  1. Minimizing redundancy.
  2. Minimizing the insertion, deletion, and update **anomalies.**
- It can be considered as a "filtering" or "purification" process to make the design have successively better quality.
- An **unsatisfactory relation schema that does not meet the condition for a normal form then that is decomposed into smaller relation schemas** that contain a subset of the attributes and meet the test for normal form.

Module – 3 Database Design Theory

Normalization of data

Thus, the normalization procedure provides database designers with the following:

1. A formal framework for analyzing relation schemas based on their keys and on the functional dependencies among their attributes
2. A series of normal form tests that can be carried out on individual relation schemas so that the relational database can be normalized to any desired degree.

Definition. The **Normal Form(NF)** of a relation refers to the highest normal form condition that it meets, and hence indicates the degree to which it has been normalized.

Module – 3 Database Design Theory

Normalization of data

Each **normalized relation should** meet the following two properties:

1. The **nonadditive join** or **lossless join** property, which **guarantees that** the spurious tuple generation problem does not occur with respect to the relation schemas created after decomposition.
2. The **dependency preservation property**, which ensures that each functional dependency is represented in some individual relation resulting after decomposition.

Note:

- The nonadditive join property is extremely critical and must be achieved at any cost.
- Whereas the dependency preservation property, although desirable, is sometimes sacrificed.

Module – 3 Database Design Theory

Denormalization

Definition: It is the process of storing the join of higher normal form relations as a base relation, which is in a lower normal form.

Definitions of Keys and Attributes Participating in Keys

Super key:

- A **superkey** of a relation schema R = {A_1, A_2, … , A_n} is a set of attributes S ⊆ R with the property **that no two tuples** t_1 and t_2 in any legal relation state r of R will have $t_1[S] = t_2[S]$.
- A key K is a superkey with the additional property that removal of any attribute from K will cause K not to be a superkey anymore.
- If a relation schema has **more than one key**, each is called a candidate key.
- One of the candidate keys is arbitrarily designated to be the **primary key**, and the others are called **secondary keys**.

Module – 3 Database Design Theory

Definition:

- An attribute of relation schema R is called a **prime attribute** of R if it is a member of some candidate key of R.
- An attribute is called **nonprime attribute** if it is not a prime attribute, that is, if it is **not a member** of any candidate key.

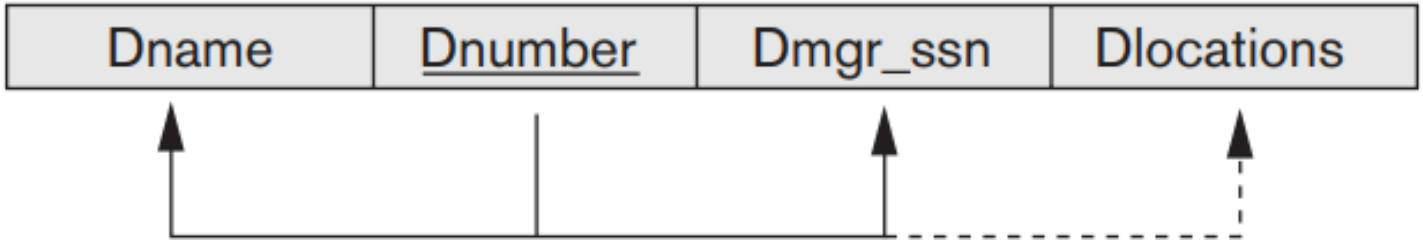
First Normal Form(1NF)

- 1NF was defined to **disallow multivalued attributes, composite attributes, and their combinations**.
- It states that the domain of an attribute must include only atomic (simple, indivisible) values and that the value of any attribute in a tuple must be a single value from the domain of that attribute.
- **Hence**, 1NF disallows having a set of values, a tuple of values, or a combination of both as an attribute value for a single tuple.

Module – 3 Database Design Theory

First Normal Form(1NF)

DEPARTMENT



DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

The relation **DEPARTMENT** is not in 1NF because Dlocations is not an atomic attribute.

Module – 3 Database Design Theory

First Normal Form(1NF)

There are **two ways we can look at the Dlocations** attribute:

1. The domain of Dlocations contains atomic values, but some tuples can have a set of these values. In this case, Dlocations is **not functionally dependent on** the primary key Dnumber.
2. The domain of Dlocations contains sets of values and hence is nonatomic. In this case, Dnumber → Dlocations because each set is considered a single member of the attribute domain.

There are **three main techniques to achieve first normal form** for such a relation:

Technique – 1:

- **Remove the attribute** Dlocations that violates 1NF and place it in a separate relation **DEPT_LOCATIONS** along with the primary key Dnumber of DEPARTMENT.

Module – 3 Database Design Theory

First Normal Form(1NF)

- The primary key of this newly formed relation is the combination {Dnumber, Dlocation}.
- A distinct tuple in **DEPT_LOCATIONS** exists for each location of a department.
- This **decomposes the non-1NF** relation into two 1NF relations.

DEPARTMENT

Dname	Dnumber	Mgr_ssn
Research	5	333445555
Administration	4	987654321
Headquarters	1	888665555

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

Module – 3 Database Design Theory

First Normal Form(1NF)

Technique – 2:

- **Expand the key** so that there will be a separate tuple in the original DEPARTMENT relation for each location of a DEPARTMENT.
- In this case, the primary key becomes the combination {Dnumber, Dlocation}.
- This solution **has the disadvantage of introducing redundancy** in the relation and hence is rarely adopted.

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocation
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

Module – 3 Database Design Theory

First Normal Form(1NF)

Technique – 3:

- If a maximum number of values is known for the attribute, for example, if it is known that at most three locations can exist for a department, then replace the Dlocations attribute by three atomic attributes: Dlocation1, Dlocation2, and Dlocation3.
- This solution has the **disadvantage of introducing NULL** values if most departments have fewer than three locations.
- It further introduces spurious semantics about the ordering among the location values; that ordering is not originally intended.

Dname	Dnumber	Dmgr_ssn	Dlocation-1	Dlocation-2	Dlocation-3

Module – 3 Database Design Theory

First Normal Form(1NF)

- First normal form also **disallows multivalued attributes that are themselves composite**.
- These are called nested relations because each tuple can have a relation within it.

EMP_PROJ(Ssn, Ename, {PROJS(Pnumber, Hours)})

- In the above schema relation PROJS(Pnumber, Hours) within each tuple represents the employee‟s projects and the hours per week that employee works on each project.

EMP_PROJ

		Projs	
Ssn	Ename	Pnumber	Hours

- **Ssn** is the primary key of the EMP_PROJ relation, whereas **Pnumber** is the partial key of the nested relation;

Module – 3 Database Design Theory

First Normal Form(1NF)

EMP_PROJ

Ssn	Ename	Pnumber	Hours
123456789	Smith, John B.	1	32.5
		2	7.5
666884444	Narayan, Ramesh K.	3	40.0
453453453	English, Joyce A.	1	20.0
		2	20.0
333445555	Wong, Franklin T.	2	10.0
		3	10.0
		10	10.0
		20	10.0
999887777	Zelaya, Alicia J.	30	30.0
		10	10.0
987987987	Jabbar, Ahmad V.	10	35.0
		30	5.0
987654321	Wallace, Jennifer S.	30	20.0
		20	15.0
888665555	Borg, James E.	20	NULL

Module – 3 Database Design Theory

First Normal Form(1NF)

To normalize the relation EMP_PROJ into 1NF,

- Remove the nested relation attributes into a new relation and propagate the primary key into it,
- The primary key of the new relation will combine the partial key with the primary key of the original relation.

EMP_PROJ1

Ssn	Ename

EMP_PROJ2

Ssn	Pnumber	Hours

Module – 3 Database Design Theory

First Normal Form(1NF)

Summary:

- **A relation will be 1NF** if it contains an atomic value. It states that an attribute of a table(relation) cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.
- First normal form disallows nested relations.

Module – 3 Database Design Theory

Full functional dependency

- A functional dependency X → Y is a **full functional dependency** if removal of any attribute **A** from **X** means that the dependency does not hold anymore;
- That is, for any attribute A ε X, (X − {A}) does not functionally determine Y.
- **Example:** {Ssn, Pnumber} → Hours is a full dependency (neither Ssn → Hours nor Pnumber → Hours holds).
- **Partial functional dependency**
- A functional dependency X → Y is a partial dependency if some attribute A ε X can be removed from X and the dependency still holds; that is, for some A ε X, (X − {A}) → Y.
- **Example:** The dependency {Ssn, Pnumber} → Ename is partial because Ssn → Ename holds (mean without Pnumber, Ename can be determined by using Ssn).

Module – 3 Database Design Theory

Second Normal Form(2NF)

- Second normal form (2NF) is based on the concept of full functional dependency.

Definition: A relation schema **R** is in **2NF** if every nonprime attribute **A** in **R** is fully functionally dependent on the primary key of R.

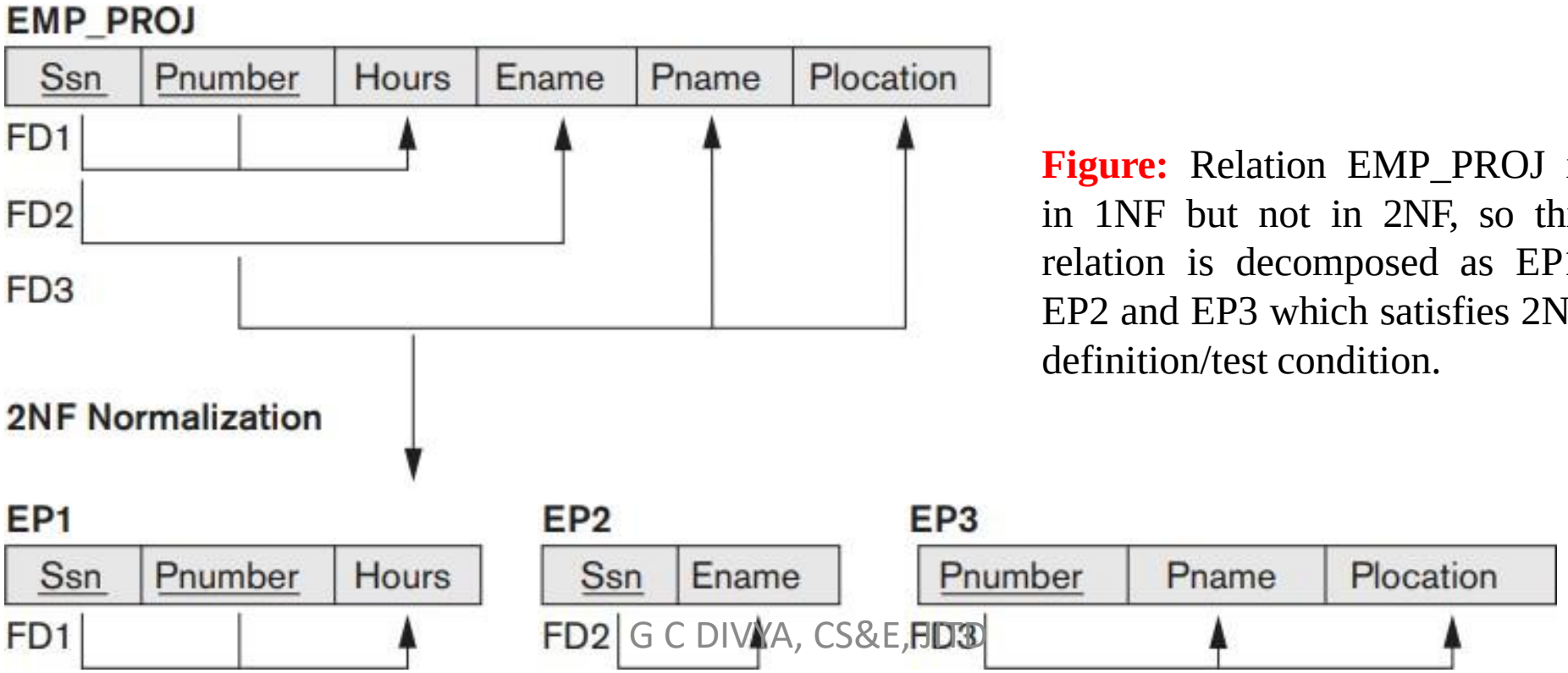


Figure: Relation EMP_PROJ is in 1NF but not in 2NF, so this relation is decomposed as EP1, EP2 and EP3 which satisfies 2NF definition/test condition.

Module – 3 Database Design Theory

Second Normal Form(2NF)

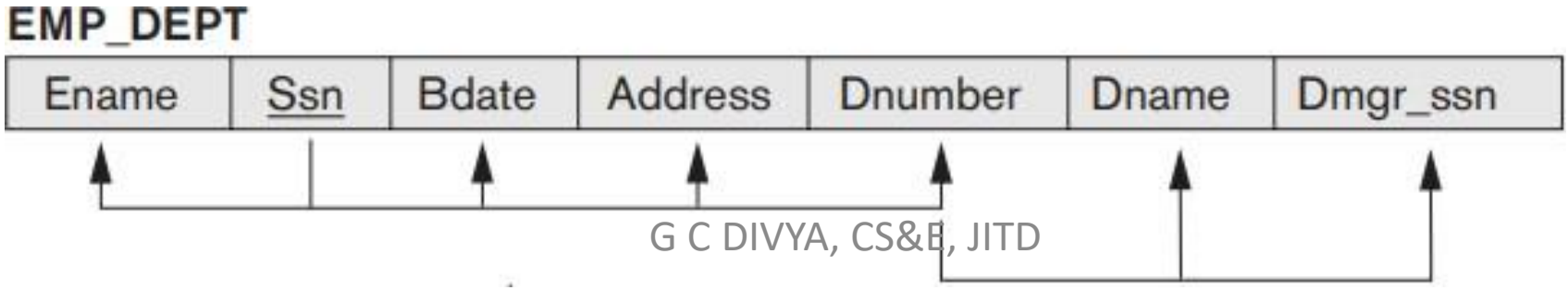
- The relation EMP_PROJ is in 1NF but not in 2NF because functional dependencies FD2 and FD3 violates 2NF, Ename can be functionally determined by only Ssn, and both Pname and Plocation can be functionally determined by only Pnumber.
- Means these three **non prime attributes**(Ename, Pname and Plocation) are not fully functionally dependent on **prime attributes**(Ssn and Pnumber) of relation EMP_PROJ.
- **To convert into 2NF**, decompose the relation such that all non prime attributes should fully dependent on prime attribute of a relation.
- **Prime attributes in DBMS are identified through primary key components and functional dependencies.**
- **A non-prime attribute instead of depending on entire Candidate key, depends on part of it.**

Module – 3 Database Design Theory

Transitive dependency

- A functional dependency **X** → **Y** in a relation schema R is a transitive dependency if there exists a set of attributes **Z** in **R** that is neither a candidate key nor a subset of any key of R, 11 and both **X** → **Z** and **Z** → Y hold.
- The dependency Ssn → Dmgr_ssn is transitive through Dnumber in EMP_DEPT.
- Means, ssn → Dnumber, Dnumber → Dmgr_SSn

Figure: Relation EMP_DEPT is not in 3NF because this has transitive functional dependency.

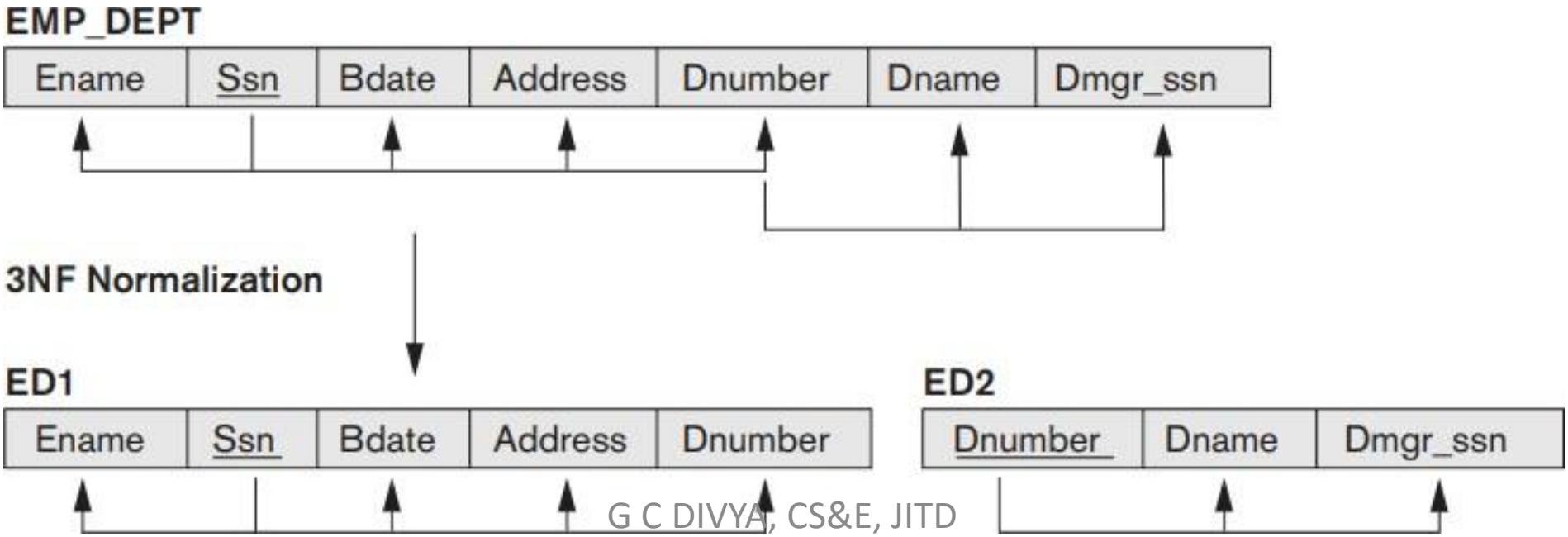


Module – 3 Database Design Theory

Third Normal Form(3NF)

- Third normal form (3NF) is based on the concept of transitive dependency.

Definition: According to Codd‟s original definition, a relation schema R is in 3NF if it satisfies 2NF and no nonprime attribute of R is transitively dependent on the primary key.



Module – 3 Database Design Theory

Summary

Normal Form	Test	Remedy (Normalization)
First (1NF)	Relation should have no multivalued attributes or nested relations.	Form new relations for each multivalued 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).

Module – 3 Database Design Theory

General definitions of 2NF and 3NF

2NF Definition: A relation schema R is in second normal form (2NF) if every nonprime attribute A in R is not partially dependent on any key of R.

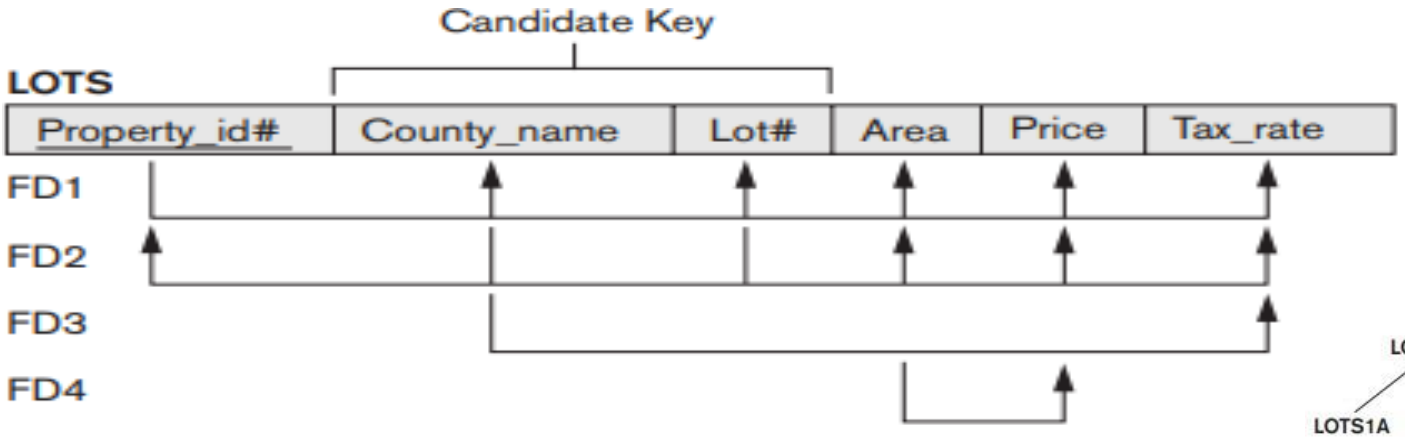
3NF Definition: A relation schema R is in third normal form (3NF) if, whenever a nontrivial functional dependency X → A holds in R, either (a) X is a superkey of R, or (b) A is a prime attribute of R.

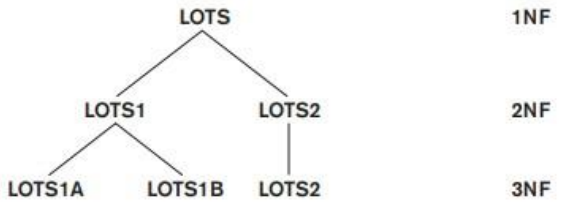
Alternative Definition of 3NF: A relation schema R is in 3NF if every nonprime attribute of R meets both of the following conditions:

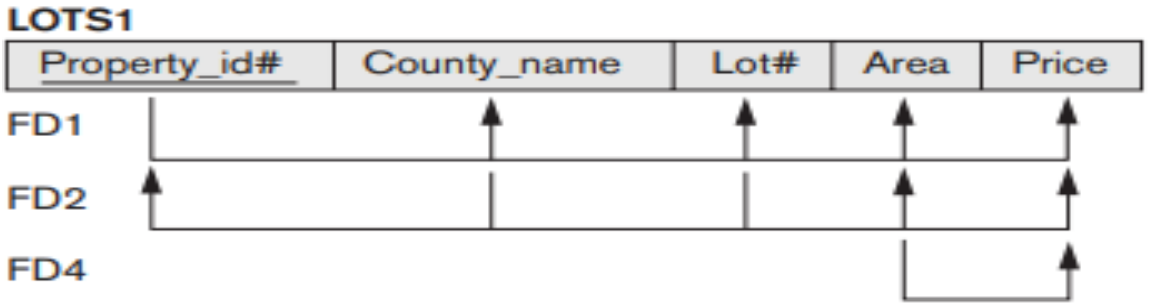
1. It is fully functionally dependent on every key of R.
2. It is non transitively dependent on every key of R.

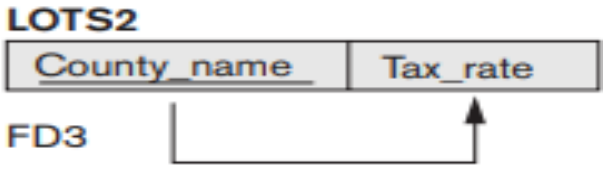
- Consider the relation **LOTS** given below…….

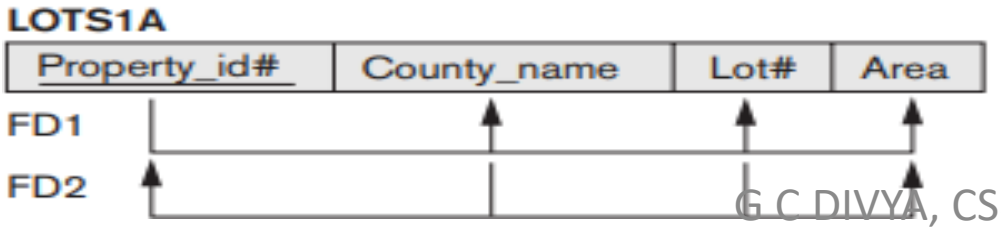
Module – 3 Database Design Theory

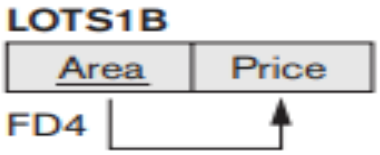












Module – 3 Database Design Theory

Boyce-Codd Normal Form

- Boyce-Codd normal form (BCNF) was proposed as a simpler form of 3NF, but it was found to be stricter than 3NF.
- That is, every relation which is in BCNF is also in 3NF, but a relation which is in 3NF is not necessarily in BCNF.
- Even when a database is in 3rd Normal Form, still there would be anomalies resulted if it has more than one **Candidate** Key.
- Sometimes BCNF is also referred as **3.5 Normal Form.**

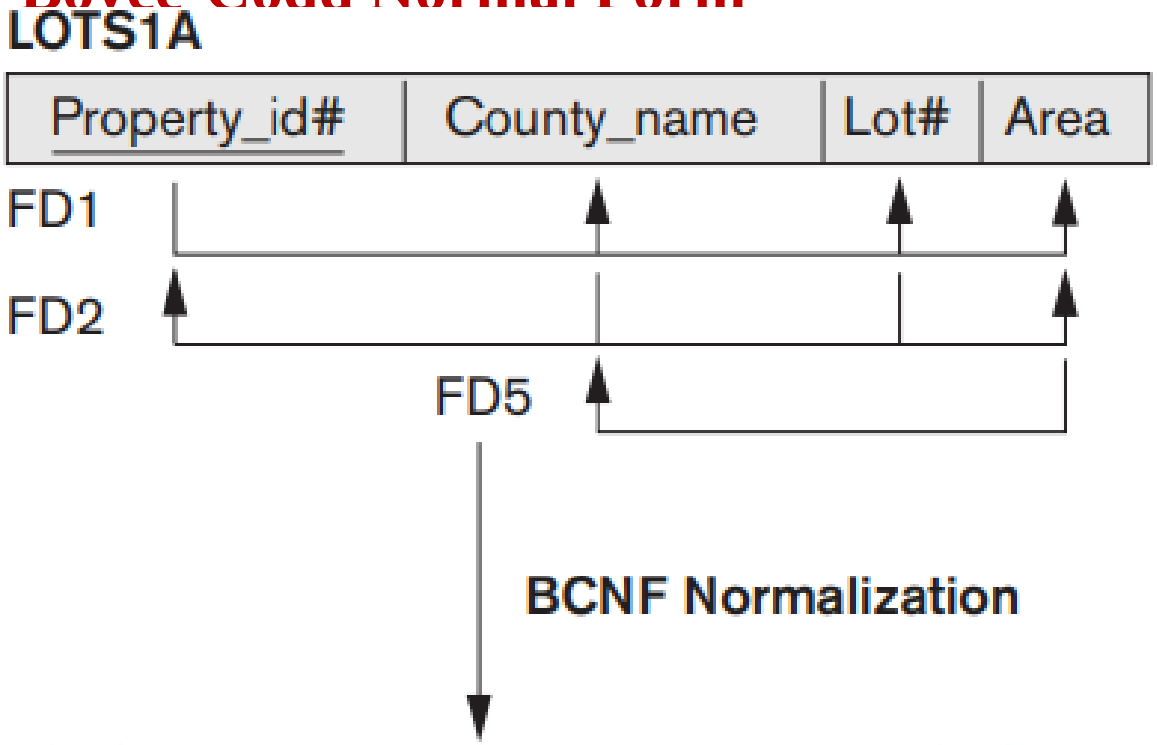
Definition. A relation schema R is in BCNF if whenever a nontrivial functional dependency **X** → **A** holds in R, then X is a superkey of R.

It states that

- BCNF is the advance version of 3NF. It is stricter than 3NF.
- A relation is in BCNF if every functional dependency X → Y, X is the super key of the relation.

Module – 3 Database Design Theory

Boyce-Codd Normal Form



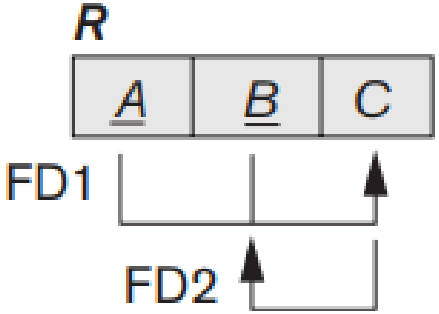


Figure: A schematic relation R with FDs; it is in 3NF, but not in BCNF due to the FD C → B. C is not a superkey of R.

Figure: BCNF normalization of LOTS1A with the functional dependency FD2 being lost in the decomposition.

Module – 3 Database Design Theory

Boyce-Codd Normal Form

- Consider the relation LOTS1A, suppose that we have thousands of lots in the relation but the lots are from only two counties: **Belthangadi** and **Ujire**.
- Suppose also that lot sizes in **Belthangadi** County are only 0.5, 0.6, 0.7, 0.8, 0.9, and 1.0 acres, whereas lot sizes in **Ujire** County are restricted to 1.1, 1.2, 1.3,1.4,1.5,1.6,1.7,1.8, 1.9, and 2.0 acres.
- In such a situation we would have the additional functional dependency FD5: **Area → County_name**. means we can determine the County name using area(since the value of area is unique from **Belthangadi and Ujire)** .
- **In relation** LOTS1A attribute area is not a key or super key, so it is not in BCNF.
- **To convert into BCNF, decomposed as** LOTS1AX and LOTS1AY.

Module – 3 Database Design Theory

Multivalued Dependencies(MVD)

- Whenever two independent 1:N relationships A:B and A:C are mixed in the same relation, R(A, B, C), an MVD may arise.

Example:

EMP

Ename	Pname	Dname
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

Definition: A multivalued dependency **X** → **Y** specified on relation schema **R**, where **X** and **Y** are both subsets of R, specifies the following constraint on any relation state r of **R**:

If two tuples t_1 and t_2 exist in r such that $t_1[X] = t_2[X]$, then two tuples t_3 and t_4 should also exist in **r** with the following properties.

Module – 3 Database Design Theory

Multivalued Dependencies(MVD)

$$t_3[X] = t_4[X] = t_1[X] = t_2[X]$$
$$t_3[Y] = t_1[Y] \text{ and } t_4[Y] = t_2[Y]$$
$$t_3[Z] = t_2[Z] \text{ and } t_4[Z] = t_1[Z]$$

where $Z = (R - (X \cup Y))$

Whenever $X \twoheadrightarrow Y$ holds, we say that X multi-determines Y.

Because of the symmetry in the definition, whenever $X \twoheadrightarrow Y$ holds in R, so does $X \twoheadrightarrow Z$. Hence, $X \twoheadrightarrow Y$ implies $X \twoheadrightarrow Z$ and therefore it is sometimes written as $X \twoheadrightarrow Y|Z$

Module – 3 Database Design Theory

Multivalued Dependencies(MVD)

Trivial MVD

An MVD X →→Y in R called a trivial MVD if Y is subset of X or X U Y = R.

Example:

EMP_PROJECTS

Ename	Pname
Smith	X
Smith	Y

EMP_DEPENDENTS

Ename	Dname
Smith	John
Smith	Anna

A trivial MVD will hold in any relation state r of R; it is called trivial because it does not specify any significant or meaningful constraint on R.

Module – 3 Database Design Theory

Multivalued Dependencies(MVD)

Nontrivial MVD

An MVD X →→Y in R called a non trivial MVD if Y is not a subset of X or X U Y != R.

Example:

EMP

Ename	Pname	Dname
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

Nontrivial MVDs in relation causes undesirable redundancy, the values „X" and „Y" of Pname are unnecessarily repeated with each value of Dname.

Note: the relations containing nontrivial MVDs tend to be all-key relations(key is all their attributes taken together).

Module – 3 Database Design Theory

Fourth normal form (4NF)

- 4NF is based on multivalued dependency, which is violated when a relation has undesirable multivalued dependencies and hence can be used to identify and decompose such relations.

Definition. A relation schema R is in 4NF with respect to a set of dependencies F (that includes functional dependencies and multivalued dependencies) if, for every nontrivial multivalued dependency $X \rightarrow\rightarrow Y$ in $\mathbf{F}^+$, X is a superkey for R.

The relation **SUPPLY** with no MVDs is in 4NF, because Sname is a superley.

SUPPLY

Sname	Part_name	Proj_name
Smith	Bolt	ProjX
Smith	Nut	ProjY
Adamsky	Bolt	ProjY
Walton	Nut	ProjZ
Adamsky	Nail	ProjX
Adamsky	Bolt	ProjX
Smith	Bolt	ProjY

Introduction to SQL

Module – 3 Introduction to SQL

Introduction

- The **relational algebra operations** are too low-level for most commercial DBMS users because a query in relational algebra is written as a sequence of operations that, when executed, produces the required result.
- Hence, the user must specify how that is, in what order to execute the query operations.
- On the other hand, the **SQL language provides a higher-level** declarative language interface, so the user only specifies **what the result is to be**, leaving the actual optimization and decisions on how to execute the query to the DBMS.
- The name **SQL** is presently expanded as **Structured Query Language**. Originally, **SQL** was called **SEQUEL** (Structured English QUEry Language) and was designed and implemented at **IBM Research** as the interface for an experimental relational database system called **SYSTEM R**.

Module – 3 Introduction to SQL

Introduction

- SQL is now the standard language for commercial relational DBMSs.
- The **standardization of SQL** is a joint effort by the **American National Standards Institute** (ANSI) and the **International Standards Organization** (ISO), and the first SQL standard is called SQL-86 or SQL1.
- SQL is a **comprehensive database language**: It has statements for data definitions, queries, and updates. Hence, it is both a **DDL** and a **DML**.
- In addition, it has facilities for defining views on the database, for specifying security and authorization, for defining integrity constraints, and for specifying transaction controls.
- It also has rules for embedding SQL statements into a general-purpose programming language such as Java or C/C++.

Module – 3 Introduction to SQL

Introduction

- SQL uses the terms **table**, **row**, and **column** for the formal relational model terms **relation**, **tuple**, and **attribute**, respectively.
- The main SQL command for data definition is the **CREATE** statement, which can be used to **create schemas, tables (relations), types, and domains, also** used to create views, assertions, and triggers.

Module – 3 Introduction to SQL

SQL Data Definition and Data Types

- The **CREATE TABLE** command is used to specify a new relation by giving it a name and specifying its attributes and initial constraints.
- The attributes are specified with a name, a data type to specify its domain of values, and constraints, such as **NOT NULL, etc**.
- The key, entity integrity, and referential integrity constraints can be specified within the **CREATE TABLE** statement after the attributes are declared, or they can be added later using the **ALTER TABLE command.**
- The relations declared through CREATE TABLE statements are called **base tables** (or base relations); this means that the table and its rows are actually created and **stored as a file** by the DBMS.

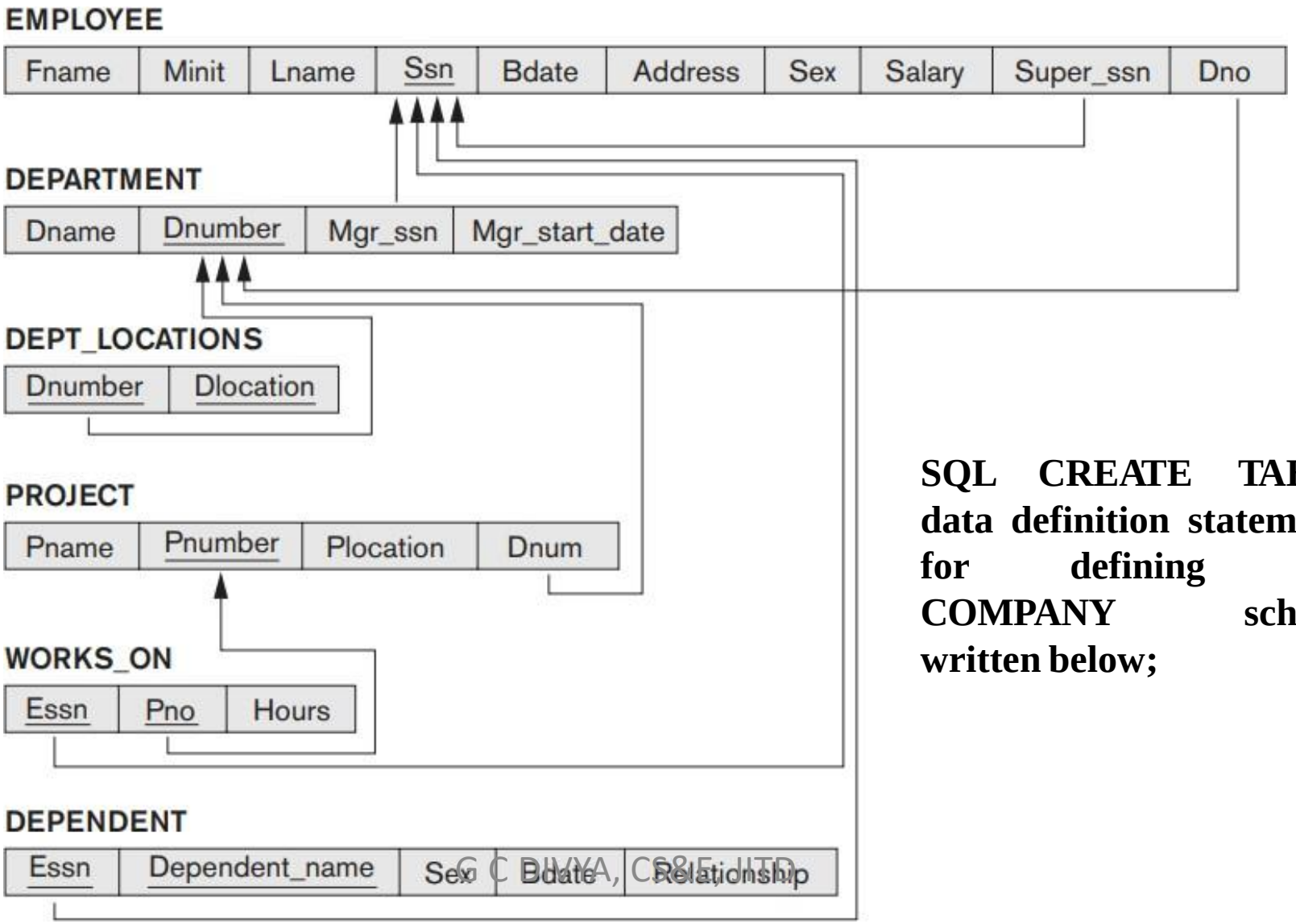
Module – 3 Introduction to SQL

SQL Data Definition and Data Types

- Base relations are distinguished from **virtual relations**, which are created through the **CREATE VIEW** statement.
- **In SQL**, the attributes in a base table are **considered to be ordered in the sequence in which they are specified in** the CREATE TABLE statement.
- However, rows (tuples) are **not considered to be ordered** within a table (relation).

Module – 3 Introduction to SQL

Referential integrity constraints displayed on the COMPANY relational database



SQL CREATE TABLE data definition statements for defining this COMPANY schema written below;

Module – 3 Introduction to SQL

SQL CREATE TABLE data definition statements for defining COMPANY schema

```
CREATE TABLE EMPLOYEE
        ( Fname          VARCHAR(15)        NOT NULL,
          Minit          CHAR,
          Lname          VARCHAR(15)        NOT NULL,
          Ssn            CHAR(9)            NOT NULL,
          Bdate          DATE,
          Address        VARCHAR(30),
          Sex            CHAR,
          Salary         DECIMAL(10,2),
          Super_ssn      CHAR(9),
          Dno            INT                NOT NULL,
        PRIMARY KEY (Ssn),
CREATE TABLE DEPARTMENT
        ( Dname          VARCHAR(15)        NOT NULL,
          Dnumber        INT                NOT NULL,
          Mgr_ssn        CHAR(9)            NOT NULL,
          Mgr_start_date DATE,
        PRIMARY KEY (Dnumber),
        UNIQUE (Dname),
        FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn) );
CREATE TABLE DEPT_LOCATIONS
        ( Dnumber        INT                NOT NULL,
          Dlocation      VARCHAR(15)        NOT NULL,
        PRIMARY KEY (Dnumber, Dlocation),
        FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber) );
```

Module – 3 Introduction to SQL

SQL CREATE TABLE data definition statements for defining COMPANY schema

```
CREATE TABLE PROJECT
        ( Pname                 VARCHAR(15)             NOT NULL,
          Pnumber               INT                     NOT NULL,
          Plocation             VARCHAR(15),
          Dnum                  INT                     NOT NULL,
        PRIMARY KEY (Pnumber),
        UNIQUE (Pname),
        FOREIGN KEY (Dnum) REFERENCES DEPARTMENT(Dnumber) );
CREATE TABLE WORKS_ON
        ( Essn                  CHAR(9)                 NOT NULL,
          Pno                   INT                     NOT NULL,
          Hours                 DECIMAL(3,1)            NOT NULL,
        PRIMARY KEY (Essn, Pno),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn),
        FOREIGN KEY (Pno) REFERENCES PROJECT(Pnumber) );
CREATE TABLE DEPENDENT
        ( Essn                  CHAR(9)                 NOT NULL,
          Dependent_name        VARCHAR(15)             NOT NULL,
          Sex                   CHAR,
          Bdate                 DATE,
          Relationship          VARCHAR(8),
        PRIMARY KEY (Essn, Dependent_name),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn) );
```

Module – 3 Introduction to SQL

Attribute Data Types and Domains in SQL

The basic data types available for attributes include

1. Numeric
2. Character string
3. Bit string
4. Boolean
5. Date and time.

In MySQL

- TINYINT = 1 byte (8 bit)
- SMALLINT = 2 bytes (16 bit)
- MEDIUMINT = 3 bytes (24 bit)
- INT = 4 bytes (32 bit)
- BIGINT = 8 bytes (64 bit).

Numeric data types:

- Integer numbers of various sizes (INTEGER or INT, and SMALLINT)
- Floating-point (real) numbers of various precision (FLOAT or REAL, and DOUBLE PRECISION).
- 4-byte for single-precision FLOAT column.
- 8-byte for double-precision DOUBLE column.

Module – 3 Introduction to SQL

Attribute Data Types and Domains in SQL

- Formatted numbers can be declared by using DECIMAL(i, j) or DEC(i, j) or NUMERIC(i, j) where i is the precision, that is the total number of decimal digits and j is the scale, that is the number of digits after the decimal point.

Character-string data types

- Fixed length: CHAR(n) or CHARACTER(n), where n is the number of characters.
- Varying length: VARCHAR(n) or CHAR VARYING(n) or CHARACTER VARYING(n), where n is the maximum number of characters.
- When specifying a literal **string value**, it is placed **between single quotation** marks (apostrophes), and it is **case sensitive**.

Module – 3 Introduction to SQL

Attribute Data Types and Domains in SQL

- For **fixed length strings**, a shorter string is padded with blank characters to the right.
- **For example,** name char(10), value „ravi“, in this case it is padded with 6 blank characters, means 6 locations right to the ravi are filled with blank character.
- **Variable-length** string data type called CHARACTER LARGE OBJECT or **CLOB** is also available to specify columns that have large text values, such as documents.
- The **CLOB** maximum length can be specified in kilobytes (K), megabytes (M), or gigabytes (G).
- **For example**, CLOB(20M) (supports in Oracle) specifies a maximum length of 20 megabytes. Data type **longtext** in Mysql.

Module – 3 Introduction to SQL

Attribute Data Types and Domains in SQL

Bit-string data types

- Fixed length BIT(n) or Varying length BIT VARYING(n), where n is the maximum number of bits.
- Literal bit strings are placed **between single quotes** but preceded by a B to distinguish them from character strings; for example, B‟10101‟.
- Another variable-length bit-string data type called **BINARY LARGE OBJECT** or BLOB is also available to specify columns that have large binary values, such as images.
- As for CLOB, the maximum length of a BLOB can be specified in kilobits (K), megabits (M), or gigabits (G).
- For example, BLOB(30G) specifies a maximum length of 30 gigabits.

Module – 3 Introduction to SQL

Attribute Data Types and Domains in SQL

A Boolean data type

- A Boolean data type has the traditional values of TRUE or FALSE.
- In SQL, because of the presence of NULL values, a three-valued logic is used, so a third possible value for a Boolean data type is **UNKNOWN**.

Date and time.

- The DATE data type has ten positions, and its components are YEAR, MONTH, and DAY in the form YYYY-MM-DD.
- The TIME data type has at least eight positions, with the components HOUR, MINUTE, and SECOND in the form HH:MM:SS.
- Only **valid dates and times** should be allowed by the SQL implementation.

Module – 3 Introduction to SQL

Attribute Data Types and Domains in SQL

- This implies that months should be between 1 and 12 and days must be between 01 and 31; furthermore, a day should be a valid day for the corresponding month.
- The < (less than) comparison can be used with dates or times, an earlier date is considered to be smaller than a later date, and similarly with time.
- Literal values are represented by **single-quoted strings** preceded by the keyword DATE or TIME;
- **for example,** DATE ‚2014-09-27‘ or TIME ‚09:12:47‘.
- The format of DATE, TIME, and TIMESTAMP can be considered as a **special type of string**.

Module – 3 Introduction to SQL

Attribute Data Types and Domains in SQL

Creating Domain

- We can specify the data type of each attribute directly.
- A domain can be declared, and this domain name can be used with the attribute specification.

CREATE DOMAIN SSN_TYPE AS CHAR(9)

- This makes it easier to change the data type for a domain that is used by numerous attributes in a schema.
- Means, instead of changing the type of all the attributes, just change the type of domain, this change reflects in all the attributes that uses this domain.

Create table employee(emp_id varchar(5), emp_ssn SSN_TYPE);

Create table Dept(dep_id varchar(5), mgr_ssn SSN_TYPE);

Module – 3 Introduction to SQL

Specifying Constraints in SQL

- Key constraints
- Referential integrity constraints
- Restrictions on attribute domains and NULLs.

Specifying Key and Referential Integrity Constraints

- Keys and referential integrity constraints are very important, there are special clauses within the **CREATE TABLE** statement to specify them.
- The **PRIMARY KEY** clause specifies one or more attributes that make up the primary key of a relation.
- If a primary key has a single attribute, the clause can follow the attribute directly.
- **For example**, **create table dept**(dept_id int auto_increment **primary key**, dname varchar (10) **unique**);

Module – 3 Introduction to SQL

Specifying Constraints in SQL

- The UNIQUE clause specifies alternate (unique) keys, **also known as candidate keys.**
- Referential integrity is specified via the **FOREIGN KEY** clause.
- **For example,**

create table employee(eid int auto_increment primary key, ename char(15),aadhar mediumint unique,dnumber int, **foreign key** (dnumber) references dept(dept_id) on delete set null);

- **Referential integrity** constraint can be violated when tuples are inserted or deleted, or when a foreign key or primary key attribute value is updated.
- The **default action** that SQL takes for an integrity violation is to reject the update/delete operation that will cause a violation, which is known as the **RESTRICT** option.

Module – 3 Introduction to SQL

Specifying Constraints in SQL

- The schema designer can specify an alternative action to be taken by attaching a **referential triggered action clause** to any foreign key constraint.
- The options include **SET NULL**, **CASCADE**, and **SET DEFAULT.**

For example for cascade on update

Create table PUBLISHER(name varchar(12),address varchar(12), phone int, primary key(name));

Create table BOOK (book_id varchar(5),title varchar(15), PUBLISHER_name varchar(10),pub_year int, primary key(book_id), **foreign key**(PUBLISHER_name) references PUBLISHER(name) **on update cascade**);

Module – 3 Introduction to SQL

Specifying Constraints in SQL

For example for cascade on delete

Create table BOOK (book_id varchar(5),title varchar(15), PUBLISHER_name varchar(10),pub_year int, primary key(book_id), **foreign key**(PUBLISHER_name) references PUBLISHER(name) **on delete cascade**);

For example for set null when record is deleted in referenced table.

Create table BOOK (book_id varchar(5),title varchar(15), PUBLISHER_name varchar(10),pub_year int, primary key(book_id), **foreign key**(PUBLISHER_name) references PUBLISHER(name) **on delete set null**);

Module – 3 Introduction to SQL

Specifying Constraints in SQL

For example for set null when record is deleted in referenced table
Create table BOOK (book_id varchar(5),title varchar(15), PUBLISHER_name varchar(10) **default "xxx",**pub_year int, primary key(book_id), **foreign key**(PUBLISHER_name) references PUBLISHER(name) **on delete set default**);

Giving Names to Constraints

- We can give a name to the constraint using the keyword CONSTRAINT. The names of all constraints within a particular schema must be unique.
- A constraint name is used to identify a particular constraint in case the constraint must be dropped later and replaced with another constraint.

Module – 3 Introduction to SQL

Specifying Constraints in SQL

- **Specifying Attribute Constraints and Attribute Defaults**
- SQL allows NULLs as attribute values.
- A constraint **NOT NULL** may be specified if NULL value is not permitted for a particular attribute of a relation.
- This is always implicitly specified for the attributes that are part of the primary key of each relation.
- But it can be specified for any other non key attributes whose values are required not to be NULL.
- It is also possible to define a default value for an attribute by appending the clause **DEFAULT** to an attribute definition.
- The default value is included in any new tuple if an explicit value is not provided for that attribute.

Module – 3 Introduction to SQL

Specifying Constraints in SQL

- Another type of constraint can restrict attribute or domain values using the **CHECK clause** following an attribute or domain definition.
- **For example,**

Create table person(id int auto_increment primary key,name varchar(10) **not null**, age int check (age between 18 and 35), location varchar(15) **default** "KAR");

Module – 3 Introduction to SQL

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

Module – 3 Introduction to SQL

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Module – 3 Introduction to SQL

Basic Retrieval Queries in SQL

- SQL has one basic statement for retrieving information from a database, that is **SELECT** statement.

The SELECT-FROM-WHERE Structure of Basic SQL Queries

The basic form of the SELECT statement, sometimes called a mapping or a select-from-where block, is formed of the three clauses SELECT, FROM, and WHERE.

General form:

SELECT <attribute list> **FROM** <table list> **WHERE**<condition> ;

where

<**attribute list**> is a list of attribute names whose values are to be retrieved by the query.

<**table list**> is a list of the relation names required to process the query.

<**condition**>is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query.

Module – 3 Introduction to SQL

Basic Retrieval Queries in SQL

- In SQL, the basic logical comparison operators for comparing attribute values with one another and with literal constants are =, <=, >, >=, and <>(not equal).
- **Example Query-1 for retrieval :** Retrieve the birth date and address of the employee(s) whose name is „John B. Smith“.

SELECT Bdate, Address **FROM** EMPLOYEE **WHERE** Fname = „John“AND Minit = „B“AND Lname = „Smith“;

- The above query involves only the EMPLOYEE relation listed in the **FROM** clause.
- The query **selects** the individual EMPLOYEE tuples that satisfy the condition of the **WHERE** clause, then projects the result on the Bdate and Address attributes listed in the SELECT clause.

Module – 3 Introduction to SQL

Basic Retrieval Queries in SQL

- **Example Query-2 for retrieval :** Retrieve the name and address of all employees who work for the „Research“ department.

SELECT Fname, Lname, Address FROM EMPLOYEE, DEPARTMENT WHERE Dname = „Research“ AND Dnumber = Dno;

- In the WHERE clause of Q2, the **condition** Dname = „Research“ is a selection condition that chooses the particular tuple of interest in the DEPARTMENT table, because Dname is an attribute of DEPARTMENT.
- The **condition** Dnumber = Dno is called a **join condition**, because it combines two tuples: one from DEPARTMENT and one from EMPLOYEE, whenever the value of Dnumber in DEPARTMENT is equal to the value of Dno in EMPLOYEE.

Module – 3 Introduction to SQL

Basic Retrieval Queries in SQL

- **Example Query-3 for retrieval:** For every project located in „Stafford", **list** the project number, the controlling department number, and the department manager"s last name, address, and birth date.

SELECT Pnumber, Dnum, Lname, Address, Bdate FROM PROJECT, DEPARTMENT, EMPLOYEE WHERE Dnum = Dnumber AND Mgr_ssn = Ssn AND Plocation = „Stafford";

Pnumber	Dnum	Lname	Address	Bdate
10	4	Wallace	291Berry, Bellaire, TX	1941-06-20
30	4	Wallace	291Berry, Bellaire, TX	1941-06-20

- A query that **involves only selection and join conditions plus projection attributes** is known as a select-project-join query.

Module – 3 Introduction to SQL

Ambiguous Attribute Names, Aliasing, Renaming, and Tuple Variables

- When the same name is used for attributes which are present in different tables and a multi-table query refers to two or more attributes with the same name, we must qualify the attribute name with the relation name to prevent ambiguity.
- This is done by prefixing the relation name to the attribute name and separating the two by a period operator(dot).

Example:

SELECT Fname, EMPLOYEE.Name, Address FROM EMPLOYEE, DEPARTMENT WHERE DEPARTMENT.Name = „Research“ AND DEPARTMENT.Dnumber = EMPLOYEE.Dnumber;

Module – 3 Introduction to SQL

Ambiguous Attribute Names, Aliasing, Renaming, and Tuple Variables

- Fully qualified attribute names can be used for clarity even if there is no ambiguity in attribute names.

For example:

SELECT EMPLOYEE.Fname, EMPLOYEE.LName, EMPLOYEE.Address FROM EMPLOYEE, DEPARTMENT WHERE DEPARTMENT.DName = „Research“ AND DEPARTMENT.Dnumber = EMPLOYEE.Dno;

- The ambiguity of attribute names also arises in the case of queries that refer to the same relation twice.
- **For example:** retrieve the employee‘s first and last name and the first and last name of his or her immediate supervisor.

Module – 3 Introduction to SQL

Ambiguous Attribute Names, Aliasing, Renaming, and Tuple Variables

SELECT E.Fname, E.Lname, S.Fname, S.Lname FROM **EMPLOYEE** AS **E**, **EMPLOYEE** AS **S** WHERE E.Super_ssn = S.Ssn;

E.Fname	E.Lname	S.Fname	S.Lname
John	Smith	Franklin	Wong
Franklin	Wong	James	Borg
Alicia	Zelaya	Jennifer	Wallace
Jennifer	Wallace	James	Borg
Ramesh	Narayan	Franklin	Wong
Joyce	English	Franklin	Wong
Ahmad	Jabbar	Jennifer	Wallace

- In the above SELECT query, we have declared alternative relation names **E** and **S**, called aliases or tuple variables, for the EMPLOYEE relation.

Module – 3 Introduction to SQL

Ambiguous Attribute Names, Aliasing, Renaming, and Tuple Variables

- It is also possible to rename the relation attributes within the query in SQL by giving them aliases.

For example:

if we write **EMPLOYEE** AS **E**(Fn, Mi, Ln, Ssn, Bd, Addr, Sex, Sal, Sssn, Dno)

- We can use this alias-naming or renaming mechanism in any SQL query to specify tuple variables for every table in the WHERE clause, whether or not the same relation needs to be referenced more than once.

For example:

SELECT E.Fname, E.LName, E.Address FROM EMPLOYEE AS E, DEPARTMENT AS D WHERE D.DName = „Research“ AND D.Dnumber = E.Dno;

Module – 3 Introduction to SQL

Unspecified WHERE Clause

- A missing **WHERE** clause indicates no condition on tuple selection; hence, all tuples of the relation specified in the **FROM** clause qualify and are selected for the query result.
- If more than one relation is specified in the **FROM** clause and there is no **WHERE** clause, then the CROSS PRODUCT, that is all possible tuple combinations of these relations is selected.

SELECT Ssn FROM EMPLOYEE;

Ssn
123456789
333445555
999887777
987654321
666884444
453453453
987987987
888665555

Module – 3 Introduction to SQL

Unspecified WHERE Clause

SELECT Ssn, Dname

FROM EMPLOYEE, DEPARTMENT;

Ssn	Dname
123456789	Research
333445555	Research
999887777	Research
987654321	Research
666884444	Research
453453453	Research
987987987	Research
888665555	Research
123456789	Administration
333445555	Administration
999887777	Administration
987654321	Administration
666884444	Administration
453453453	Administration
987987987	Administration
888665555	Administration
123456789	Headquarters
333445555	Headquarters
999887777	Headquarters
987654321	Headquarters
666884444	Headquarters
453453453	Headquarters
987987987	Headquarters
888665555	Headquarters

Module – 3 Introduction to SQL

Use of the Asterisk

- To retrieve all the attribute values of the selected tuples, we do not have to list the attribute names explicitly in SQL; we **just specify an asterisk (*),** which stands for all the attributes.

SELECT * FROM EMPLOYEE **WHERE** Dno = 5;

The above query retrieves all the attribute values of any EMPLOYEE who works in DEPARTMENT number 5.

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-09-01	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

Module – 3 Introduction to SQL

Use of the Asterisk

Select *from publisher;

name	address	phone
Karnataka	Bengaluru	9741970005
McGraw	Delhi	9845098450
Sapna	Bengaluru	9741970005
Weilly	Bengaluru	9741970005

select *from book;

book_id	title	PUBLISHER_name	pub_year
123	Python	Sapna	2020
127	DBMS	Weilly	2021
128	ML	Sapna	2021

select *from publisher,book **where** publisher_name = name;

- The above query selects all the attributes of tuple from both tables which satisfies the condtion in WHERE clause.

name	address	phone	book_id	title	PUBLISHER_name	pub_year
Sapna	Bengaluru	9741970005	123	Python	Sapna	2020
Weilly	Bengaluru	9741970005	127	DBMS	Weilly	2021
Sapna	Bengaluru	9741970005	128	ML	Sapna	2021

Module – 3 Introduction to SQL

Use of the Asterisk

Select *from publisher,book;

The above query selects all attributes and all tuple from both tables which results in **CROSS PRODUCT**(since no WHERE clause).

name	address	phone	book_id	title	PUBLISHER_name	pub_year
Weilly	Bengaluru	9741970005	123	Python	Sapna	2020
Sapna	Bengaluru	9741970005	123	Python	Sapna	2020
McGraw	Delhi	9845098450	123	Python	Sapna	2020
Karnataka	Bengaluru	9741970005	123	Python	Sapna	2020
Weilly	Bengaluru	9741970005	127	DBMS	Weilly	2021
Sapna	Bengaluru	9741970005	127	DBMS	Weilly	2021
McGraw	Delhi	9845098450	127	DBMS	Weilly	2021
Karnataka	Bengaluru	9741970005	127	DBMS	Weilly	2021
Weilly	Bengaluru	9741970005	128	ML	Sapna	2021
Sapna	Bengaluru	9741970005	128	ML	Sapna	2021
McGraw	Delhi	9845098450	128	ML	Sapna	2021
Karnataka	Bengaluru	9741970005	128	ML	Sapna	2021

Module – 3 Introduction to SQL

Tables as Sets in SQL

- SQL usually treats a table not as a set but rather as a multiset, means duplicate tuples can appear more than once in a table, and in the result of a query.
- SQL does not automatically eliminate duplicate tuples in the results of queries, for the **following reasons**:
  1. Duplicate elimination is an expensive operation. One way to implement it is to sort the tuples first and then eliminate duplicates.
  2. The user may want to see duplicate tuples in the result of a query.

- An SQL table with a key is restricted to being a set(unique tuples), since the key value must be distinct in each tuple.

Module – 3 Introduction to Databases

Tables as Sets in SQL

- If we want to eliminate duplicate tuples from the result of an SQL query, we have to use the keyword **DISTINCT** in the SELECT clause.
- A query with **SELECT DISTINCT** eliminates duplicates, whereas a query with **SELECT ALL** keeps duplicates.
- **Query:** Retrieve the salary of every employee

SELECT ALL Salary FROM EMPLOYEE;

If several employees have the same salary, that salary value will appear as many times in the result of the query.

Salary
30000
40000
25000
43000
38000
25000
25000
55000

Module – 3 Introduction to SQL

Tables as Sets in SQL

- If we are interested only in **distinct salary values**, we want each value to appear only once, then

Query to retrieve all distinct salary values.

```
SELECT DISTINCT Salary FROM EMPLOYEE;
```

Salary
30000
40000
25000
43000
38000
55000

Module – 3 Introduction to SQL

Tables as Sets in SQL

- SQL has directly incorporated some of the set operations from mathematical set theory, There are
  1. set union (UNION).
  2. set difference (EXCEPT).
  3. set intersection (INTERSECT) operations.

- The relations resulting from these set operations are sets of tuples; that is, duplicate tuples are eliminated from the result.
- These set operations apply only to **type-compatible** relations.
- So we must make sure that the two relations on which we apply the operation have the same attributes and that the attributes appear in the same order in both relations.

Module – 3 Introduction to SQL

Tables as Sets in SQL

Example of SQL UNION operation

Query: Make a list of all project numbers for projects that involve an employee whose last name is „Smith“, either as a worker or as a manager of the department that controls the project.

(SELECT DISTINCT Pnumber FROM PROJECT, DEPARTMENT, EMPLOYEE WHERE Dnum = Dnumber AND Mgr_ssn = Ssn AND Lname = „Smith“)

UNION

(SELECT DISTINCT Pnumber FROM PROJECT, WORKS_ON, EMPLOYEE WHERE Pnumber = Pno AND Essn = Ssn AND Lname = „Smith“);

Module – 3 Introduction to SQL

Tables as Sets in SQL

In the above query;

- The first SELECT query retrieves the projects that involve a „Smith" as manager of the department that controls the project.
- The second SELECT query retrieves the projects that involve a „Smith" as a worker on the project.
- If several employees have the last name „Smith", the project names involving any of them will be retrieved.
- Applying the UNION operation to the two SELECT queries gives the unique project numbers.

Module – 3 Introduction to SQL

Tables as Sets in SQL

- SQL also has corresponding multiset operations, which are followed by the keyword ALL.
  1. UNION ALL,
  2. EXCEPT ALL,
  3. INTERSECT ALL).

Their results are multisets (duplicates are not eliminated).

R

A
a1
a2
a2
a3

S

A
a1
a2
a4
a5

Two tables, R(A) and S(A).

R(A) UNION ALL S(A).

A
a1
a1
a2
a2
a2
a3
a4
a5

Module – 3 Introduction to SQL

Tables as Sets in SQL

R

A
a1
a2
a2
a3

S

A
a1
a2
a4
a5

Two tables, R(A) and S(A).

R(A) EXCEPT ALL S(A).

A
a2
a3

R(A) INTERSECT ALL S(A).

A
a1
a2

Module – 3 Introduction to SQL

Tables as Sets in SQL

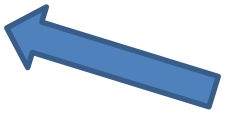
Salesman

sid	sname	city	commission
bb003	umesh	mumbai	5500
bl002	Ramesh	bengaluru	3500
dl004	parmesh	delhi	5000
hs001			0
hy005	reddy	hyderabad	3000
ml001	suresh	mangaluru	2500

Customer

cid	cname	city	grade	sid
mrc01	subhash	mumbai	3	bb003
mrc02	abhilash	mumbai	3	bb003
mrc03	mahesh	bengaluru	4	bl002
mrc04	rajesh	bengaluru	4	bl002
mrc05	brijesh	bengaluru	2	bl002
mrc06	sundresh	bengaluru	1	bl002
mrc07	rudresh	bengaluru	1	bl002
mrc08	rudresh	delhi	2	dl004
mrc09	rudresh	delhi	3	dl004
mrc10	milana	mangaluru	3	ml001
mrc11	james	mangaluru	3	ml001
mrc12	mary	mangaluru	3	ml001
mrc15	subhash	mumbai	3	hy005

sname	same_city
umesh	exists
umesh	exists
Ramesh	exists
Ramesh	exists
Ramesh	exists
Ramesh	exists
Ramesh	exists
parmesh	exists
parmesh	exists
suresh	exists
suresh	exists
suresh	exists



```
select sname,'exists'as same_city from salesman s,customer c where s.city =c.city and s.sid = c.sid;
```

Module – 3 Introduction to SQL

Tables as Sets in SQL

select sname,'not exists,, as same_city from salesman s,customer c where s.city != c.city and s.sid = c.sid;

sname	same_city
reddy	not exists

(select sname,'not exists'as same_city from salesman s,customer c where s.city != c.city and s.sid = c.sid) **union** (select sname, 'exists' as same_city from salesman s, customer c where s.city = c.city and s.sid = c.sid);

sname	same_city
reddy	not exists
umesh	exists
Ramesh	exists
parmesh	exists
suresh	exists

Module – 3 Introduction to SQL

Substring Pattern Matching and Arithmetic Operators

Pattern matching: SQL allows **two features** for pattern matching

1. Using the **LIKE** keyword as a comparison operator which finds the substring / part of a character string from the value of a particular attribute we specify in the query.

- Partial strings are specified using two reserved characters: **%** replaces an arbitrary number of zero or more characters and the underscore(_) replaces a single character..

Example-1:

Retrieve all employees whose address is in Houston, Texas.

SELECT Fname, Lname FROM EMPLOYEE WHERE Address LIKE '%Houston,TX%';

Module – 3 Introduction to SQL

Substring Pattern Matching and Arithmetic Operators

Example-2: retrieve all the customer details who are from city Bengaluru and Mangaluru.

select *from customer where city like '%gal%';

cid	cname	city	grade	sid
mrc03	mahesh	bengaluru	4	bl002
mrc04	rajesh	bengaluru	4	bl002
mrc05	brijesh	bengaluru	2	bl002
mrc06	sundresh	bengaluru	1	bl002
mrc07	rudresh	bengaluru	1	bl002
mrc10	milana	mangaluru	3	ml001
mrc11	james	mangaluru	3	ml001
mrc12	mary	mangaluru	3	ml001

Module – 3 Introduction to SQL

Substring Pattern Matching and Arithmetic Operators

Book_lending

Book_id	Branch_id	Card_no	Date_out	Due_date
121	SDM01	su003	2023-11-09	2023-11-23
123	SDM01	SU001	2024-01-02	2024-01-17
123	SDM01	su004	2023-12-09	2023-12-23
127	SDM01	su001	2024-01-02	2024-01-11
127	SDM02	su10	2024-02-07	2024-02-22

Example-3: retrieve the details of books which are issued in the year 2023.

select *from book_lending where date_out like '2023_____';

Book_id	Branch_id	Card_no	Date_out	Due_date
121	SDM01	su003	2023-11-09	2023-11-23
123	SDM01	su004	2023-12-09	2023-12-23

Module – 3 Introduction to SQL

Substring Pattern Matching and Arithmetic Operators

2. **Another feature** allows the use of arithmetic in queries. The standard arithmetic operators for addition (+), subtraction (−), multiplication (*), and division (/) can be applied to numeric values or attributes with numeric domains.

Example - 1 : raise the commission of salesmans‟ by 10% who work in city Bengaluru and Mangaluru.

select sname, 1.1 * commission as incresed_commission from salesman where city like '%gal%';

sname	incresed_commission
Ramesh	3850.0
suresh	2750.0

Module – 3 Introduction to SQL

Substring Pattern Matching and Arithmetic Operators

Example - 2 : raise the salary by 10% for the employees who works on the project **ProductX.**

SELECT E.Fname, E.Lname, 1.1 * E.Salary AS Increased_sal FROM EMPLOYEE AS E, WORKS_ON AS W, PROJECT AS P WHERE E.Ssn = W.Essn AND W.Pno = P.Pnumber AND P.Pname = 'ProductX';

- **For string data types**, the concatenate operator || can be used in a query to append two string values.
- **For date, time, timestamp**, and interval data types, operators include incrementing (+) or decrementing (−) a date, time, or timestamp by an interval.

Module – 3 Introduction to SQL

Substring Pattern Matching and Arithmetic Operators

- **Another comparison operator** is BETWEEN, which can be used as per your requirement.

Example – 1: Retrieve all employees in department 5 whose salary is between $30,000 and $40,000.

SELECT * FROM EMPLOYEE WHERE (Salary BETWEEN 30000 AND 40000) AND Dno = 5;

(Salary BETWEEN 30000 AND 40000) is Equivalent to the condition ((Salary >= 30000) AND (Salary <= 40000))

Example – 2: retrieve all the books whose id between 121 and 125.

select *from book where book_id between 121 and 125;

book_id	title	PUBLISHER_name	pub_year
121	DBMS	Weilly	2020
123	Python	Sapna	2020

Module – 3 Introduction to SQL

Ordering of Query Results

- SQL allows the user to **order the tuples in the result** of a query by the values of one or more of the attributes that appear in the query result, by using the **ORDER BY** clause.

General form

- SELECT <attribute list> FROM <table list> [WHERE <condition>] [ORDER BY<attribute list]

Example – 1: Retrieve a list of employees and the projects they are working on, ordered by department and, within each department, ordered alphabetically by last name, then first name.

SELECT D.Dname, E.Lname, E.Fname, P.Pname FROM DEPARTMENT AS D, EMPLOYEE AS E, WORKS_ON AS W, PROJECT AS P WHERE D.Dnumber = E.Dno AND E.Ssn = W.Essn AND W.Pno = P.Pnumber ORDER BY D.Dname, E.Lname, E.Fname;

Module – 3 Introduction to SQL

INSERT, DELETE, and UPDATE Statements in SQL

The Insert command

- **First form** of INSERT is used to add a single tuple (row) to a relation (table).
- We must specify the relation name and a list of values for the tuple.

Example:

INSERT INTO EMPLOYEE VALUES („Richard“, „K“, „Marini“, „653298653“, „1962-12-30“, „98 Oak Forest, Katy, TX“, „M“, 37000, „653298653“, 4);

- **A second form** of the INSERT statement allows the user to specify explicit attribute names that correspond to the values provided in the INSERT command.
- This is useful if a relation has many attributes but only a few of those attributes are assigned values in the new tuple.
- The values must include all attributes with **NOT NULL** specification and **no default value.**

Module – 3 Introduction to SQL

INSERT, DELETE, and UPDATE Statements in SQL

Example:

INSERT INTO EMPLOYEE (Fname, Lname, Dno, Ssn) VALUES („Richard“, „Marini“, 4, „653298653“);

- **Third form** of using insert, it is also possible to insert into a relation multiple tuples separated by commas in a single INSERT command.

Example:

```
MySQL  localhost:33060+ ssl  comp  SQL > insert into person(name,age) values('ravi',32),('raj',30);
Query OK, 2 rows affected (0.0098 sec)

Records: 2  Duplicates: 0  Warnings: 0
MySQL  localhost:33060+ ssl  comp  SQL > select *from person;
+----+------+-----+----------+
| id | name | age | location |
+----+------+-----+----------+
|  1 | ravi |  30 | KAR      |
|  2 | ravi |  35 | KAR      |
|  3 | ravi |  32 | KAR      |
|  4 | raj  |  30 | KAR      |
+----+------+-----+----------+
4 rows in set (0.0008 sec)

MySQL  localhost:33060+ ssl  comp  SQL > desc person;
+----------+-------------+------+-----+---------+----------------+
| Field    | Type        | Null | Key | Default | Extra          |
+----------+-------------+------+-----+---------+----------------+
| id       | int         | NO   | PRI | NULL    | auto_increment |
| name     | varchar(10) | NO   |     | NULL    |                |
| age      | int         | YES  |     | NULL    |                |
| location | varchar(15) | YES  |     | KAR     |                |
+----------+-------------+------+-----+---------+----------------+
4 rows in set (0.0058 sec)
```

Module – 3 Introduction to SQL

INSERT, DELETE, and UPDATE Statements in SQL

- **Fourth form** of the INSERT command inserts multiple tuples into a relation in conjunction with creating the relation and loading it with the result of a query.

Example:

CREATE TABLE WORKS_ON_INFO (Emp_name VARCHAR(15), Proj_name VARCHAR(15), Hours_per_week DECIMAL(3,1));

INSERT INTO WORKS_ON_INFO (Emp_name, Proj_name, Hours_per_week) SELECT E.Lname, P.Pname, W.Hours FROM PROJECT P, WORKS_ON W, EMPLOYEE E WHERE P.Pnumber = W.Pno AND W.Essn = E.Ssn;

Module – 3 Introduction to SQL

INSERT, DELETE, and UPDATE Statements in SQL

The DELETE Command

- The DELETE command removes tuples from a relation.
- It includes a **WHERE** clause to select the tuples to be deleted.
- Tuples are explicitly deleted from only one table at a time.
- Depending on the number of tuples selected by the condition in the WHERE clause, zero, one, or several tuples can be deleted by a single DELETE command.
- If **WHERE** clause is **not used** in delete statement which specifies that **all tuples in the relation are to be deleted**; however, the table remains in the database as an empty table.

Delete from table_name // deletes all tuples

Delete from table_name where condition // deletes only tuples which matches the condition.

Module – 3 Introduction to SQL

INSERT, DELETE, and UPDATE Statements in SQL

The UPDATE Command

- The UPDATE command is used to modify attribute values of one or more selected tuples.
- A **WHERE** clause in the UPDATE command selects the tuples to be modified from a single relation.
- An additional **SET** clause is used in the UPDATE command to specify the attributes to be modified and their new values.

Example:

UPDATE PROJECT SET Plocation = „Bellaire“, Dnum = 5 WHERE Pnumber = 10;

Module – 3 Introduction to SQL

INSERT, DELETE, and UPDATE Statements in SQL

The UPDATE Command

- Several tuples can also be modified with a single UPDATE command.

Example:

Give 10% raise in salary to all employees in the „Research" department.

```
UPDATE EMPLOYEE SET Salary = Salary * 1.1 WHERE Dno = 5;
```

Module – 3 Introduction to SQL

Schema Change Statements in SQL

- The schema evolution commands available in SQL, which can be used to alter a schema by adding or dropping tables, attributes, constraints, and other schema elements.

The Drop command

- The DROP command can be used to drop named schema elements, such as tables, domains, types, or constraints.
- You can also drop a whole schema if it is no longer needed by using the DROP SCHEMA command.
- There are two drop behavior options: **CASCADE** and **RESTRICT**.
- To remove the COMPANY database schema and all its tables, domains, and other elements, the CASCADE option is used as follows:

DROP SCHEMA COMPANY CASCADE;

Module – 3 Introduction to SQL

Schema Change Statements in SQL

- If the RESTRICT option is chosen in place of CASCADE, the schema is dropped only if it has no elements in it; otherwise, the DROP command will not be executed.
- If a **base relation** within a schema is no longer needed, the relation and its definition can be deleted by using the **DROP TABLE** command.

Note: the DROP TABLE command not only deletes all the records in the table if successful, but also removes the table definition from the catalog.

Module – 3 Introduction to SQL

Schema Change Statements in SQL

The ALTER Command

- The definition of a base table or of other named schema elements can be changed by using the ALTER command.
- **For base tables,** using alter table command we can
  - Add or drop a column (attribute),
  - change a column definition
  - add or drop table constraints.

- **For adding new column**

  ALTER TABLE *table_name* ADD *column_name datatype*;

- **For dropping column**

  ALTER TABLE *table_name* DROP COLUMN *column_name*;

Module – 3 Introduction to SQL

Schema Change Statements in SQL

The ALTER Command

- **For modifying the data type of the attribute**

  ALTER TABLE *table_name* MODIFY *column_name datatype*;

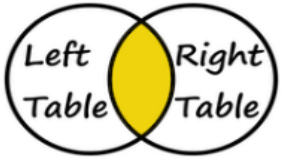
- **For renaming the column**

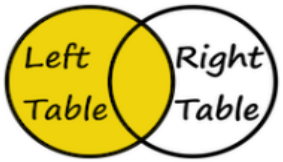
ALTER TABLE table_name RENAME COLUMN old_name to new_name

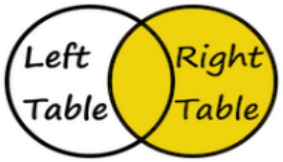
Source

Text books

1. Fundamentals of Database Systems, Ramez Elmasri and Shamkant B. Navathe, 7th Edition, 2017, Pearson.
2. Database management systems, Ramakrishnan, and Gehrke, 3rd Edition, 2014, McGraw Hill

Join Type	Description
 `INNER JOIN`	Returns rows when there is at least one row in both tables that match the join condition.


`LEFT OUTER JOIN`
or
`LEFT JOIN`
	Returns rows that have data in the left table (left of the `JOIN` keyword), even if there's no matching rows in the right table.


`RIGHT OUTER JOIN`
or
`RIGHT JOIN`
	Returns rows that have data in the right table (right of the `JOIN` keyword), even if there's no matching rows in the left table.

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	XXXXXXXXXX	18
2	PRATIK	BIHAR	XXXXXXXXXX	19
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

SELECT s.roll_no, s.name, s.address, s.phone, s.age, sc.course_id
FROM Student s
JOIN StudentCourse sc **ON** s.roll_no = sc.roll_no;

ROLL_NO	NAME	ADDRESS	PHONE	AGE	COURSE_ID
1	HARSH	DELHI	XXXXXXXXXX	18	1
2	PRATIK	BIHAR	XXXXXXXXXX	19	2
3	RIYANKA	SILGURI	XXXXXXXXXX	20	2
4	DEEP	RAMNAGAR	XXXXXXXXXX	18	3
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19	1

INNER JOIN Example

SELECT StudentCourse. COURSE_ID, Student.NAME, Student.AGE **FROM** Student
INNER JOIN StudentCourse
ON Student.ROLL_NO = StudentCourse.ROLL_NO;

COURSE_ID	NAME	Age
1	HARSH	18
2	PRATIK	19
2	RIYANKA	20
3	DEEP	18
1	SAPTARHI	19

LEFT JOIN

```
SELECT Student.NAME, StudentCourse.COURSE_ID
FROM Student
LEFT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	*NULL*
ROHIT	*NULL*
NIRAJ	*NULL*

RIGHT JOIN Example

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
RIGHT JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
NULL	4
NULL	5
NULL	4

FULL JOIN

```
SELECT Student.NAME,StudentCourse.COURSE_ID
FROM Student
FULL JOIN StudentCourse
ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	NULL
ROHIT	NULL
NIRAJ	NULL
NULL	4
NULL	5
NULL	4