20MCA102: Advanced Database Management Systems

A Simple Approach to DBMS

Dr. Geevar C Zacharias

First electronic publishing, October 2018

Contents

Preface

In an electronic world, the users are interacting with a wide range of services through different applications. It is important that, these application should store data about various entities for its proper functioning. Database is one of the important tool to store and retrieve these data. Today, the database management has evolved from a specialized computer application to a central component of a modern computing environment, and, as a result, knowledge about database systems has become an essential part of an education in computer science.

The object of this course is to develop and manage efficient and effective database applications that requires understanding the fundamentals of database management systems, techniques for the design of databases, and principles of database administration.

Being the importance of learning the technology and principles of database management system, it is learned that understanding the concepts of database technologies is a bit complex. Therefore, this book is intended for giving a simplified approach to learn database management systems, by replacing the lengthy and complex details of these concepts with a more simplified and precise explanations.

This book is organized into different modules that corresponds with the syllabus prescribed by the Kerala Technological University (KTU) for the subject 20MCA102: Advanced Database Management Systems. This material is not a replacement of any of the database related book, rather can only be viewed as a supplementary material. The content of the book is not verified by the author thoroughly. So mistakes if any, may kindly be brought into the notice of the author.

Thank you.

Module - I

Introduction

Q1. Explain the different advantageous of database.

Keeping organizational information in a file-processing system has a number of major disadvantages:

Data redundancy and inconsistency can be avoided: Since different application programs are created over a long period for managing the same data, the same information may be duplicated in several places (files). For example, an employee details may be repeated in the department file and project file. This redundancy leads to higher storage and access cost. It may also lead to data inconsistency. That is, changing the address of an employee in the department file but not in the project file.

Difficulty in accessing data: In file-processing system, the data are accessed through the functions written in the program. Thus, new functions should be written to meet the present demand. Therefore, the conventional file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner.

Data isolation: Because the data are stored in various files and have different formats, writing new application programs to retrieve the appropriate data is difficult.

Integrity problems: The data values stored in the database may need to satisfy certain consistency constraints. As there in automatic mechanism to ensure this consistency in the file-processing system, the developer need to enforce these constraints in the system by adding appropriate code in the various application programs.

Atomicity problems: In many applications, it is crucial that the data to be restored to the consistent state after a failure. So in database applications, every transactions must happen in its entirety or not at all. That is, all the transactions must be atomic. It is difficult to ensure atomicity in a conventional file-processing system.

Concurrent-access anomalies: For the overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously. To protect the consistency among data while accessing concurrently, proper monitoring functions need to written and this is difficult in the conventional file-processing system.

Security problems: Not every user of the database system should be able to access all the data. But, since application programs are added to the file-processing system in an adhoc manner, enforcing such security constraints is difficult.

Q2. Describe Three Levels of data abstraction.

Physical level: The lowest level of abstraction describes how the data are actually stored. The physical level describes complex low-level data structures in detail.

Logical level: This abstraction describes what data are stored in the database, and what relationships exist among those data.

View level: The highest level of abstraction describes only part of the entire database. The view level of abstraction exists to simplify the user interaction with the system. The system may provide many views for the same database.

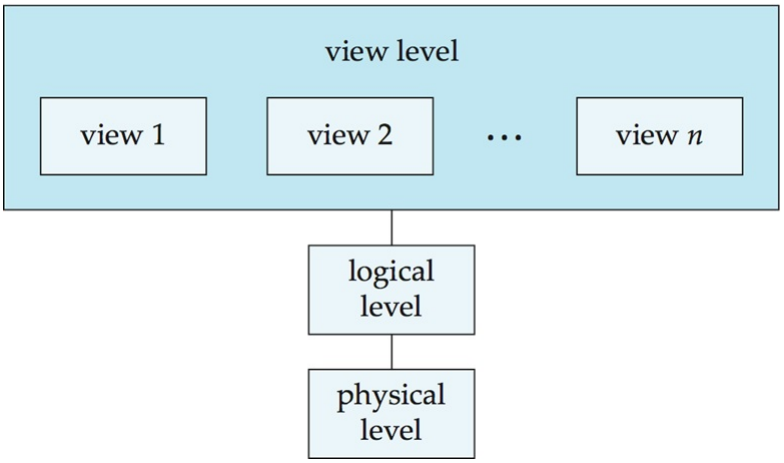


Figure 1: The three levels of data abstraction

Q3. What do you mean by data independence? Define two types of data independence in DBMS.

The capacity to change the schema at one level of a database system without having to change the schema at the next higher level is known as data independence. There are two types of data independence:

Logical data independence is the is the ability to modify the conceptual scheme without making it necessary to change external schema or rewrite application programs. Such a modification might be adding a field to a record, to change constraint, or to reduce the database by removing data item.

Physical data independence is the ability to modify the physical/internal scheme without making it necessary to change the conceptual schema. Such modifications include changing from unblocked to blocked record storage, or from sequential to random access files.

Q4. Explain the different data models.

The data models can be classified into four different categories:

Relational Model: The relational model uses collection of tables of represent both data and the relationships among those data. Each table has multiple columns, and each column has a

unique name. Tables are also known as relations. Each table contains records of a particular type. Each record type defines a fixed number of fields, or attributes. The columns of the table correspond to the attributes of the record type.

Entity-Relationship Model: The entity-relationship (E-R) data model uses a collection of basic objects, called entities and relationships among these objects. An entity is a "thing" or "object" in the real world that is distinguishable from other objects.

Object-Based Data Model: Object-Based data model can be viewed as an extension of the E-R model with notions of encapsulation, methods (functions) and object identity. The object-relational data model combines features of the object-oriented data model and relational data model.

Semistructured Data Model: The semistructured data model permits the specification of data where individual data items of the same type may have different sets of attributes.

Q5. Explain the different database languages.

A database system provides a data-definition languages to specify the database schema and a data-manipulation language to express database queries and updates.

Data-Manipulation Languages: A data-manipulation language (DML) is a language that enables users to access or manipulate data as organized by the appropriate data model. The type of access are:

- Retrieval of information stored in the database
- Insertion of new information into the database
- Deletion of information from the database
- Modification of information stored in the database

There are basically two types:

Procedural DMLs require a user to specify what data are needed and how to get those data.

Declarative DMLs (also referred as a nonprocedural DMLs) require a user to specify what data are needed without specifying how to get those data.

Data-Definition Language: Data-definition language (DDL) is used to specify the following:

- specify a database schema by a set of definitions
- specify the storage structure and access method used by the database system
- provide certain consistency constraints like:
  - Domain constraints
  - Referential Integrity
  - Assertions
  - Authorization

Q6. Explain different types database users.

There are four different types of database-system users, differentiated by the way they expect to interact with the system. Different type of user interfaces have been designated for the different types of users.

Naive users: are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously. The typical user interface for naive users is a form interface, where the users can fill in appropriate fields of the form. Naive users may also read reports generated from the database.

Application programmers: are computer professionals who develop application programs. Application programmers can choose from many tools to develop user interface.

Sophisticated users: interact with the system without writing programs. Instead, they form their requests either using a database query language or by using tools such as data analysis software.

Specialized users: are sophisticated users who write specialized database applications that do not fit into traditional data-processing framework such as Knowledge based system, expert systems etc.

Q7. Explain the functions of database administrators.

The functions of a database administrators (DBAs) include:

Schema definition: The DBS creates the original database schema by executing a set of data definition statements in the DDL.

Storage structure and access method definition

Schema and physical organization modification: The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization, or to alter the physical organization to improve performance.

Granting of authorization for data access: Granting different types of authorization for data access to various users.

Routine maintenance: Maintenance activities are:

- Perform periodical backup of the database.
- Ensuring that enough free space disk space is available for applications and upgrading disk space if required.
- Monitoring jobs running on the database and ensuring that performance is not degraded.

Entity Relationship Data Model

Q8. Explain different types of attributes in E-R model?

Attributes are a mechanism to specify the different properties of an Entity type. Several types of attributes occur in the E-R model. They are:

Composite versus Simple attributes: Composite attributes can be divided into smaller subparts, which represent more basic attributes with independent meanings. For example, the 'address' attribute of the EMPLOYEE entity can be subdivided into 'street, city, state and zip'.

Attributes that are not divisible are called Simple or atomic attributes.

Single valued versus Multivalued attributes: Most attributes have a single value for a particular entity; such attributes are called single valued. For example, 'age' is a single valued attribute of a person.

In some cases, attributes can have a set of values for the same entity - For example, a 'color' attribute for a car. Such attributes are called multivalued.

Stored versus Derived attributes: In some cases, two (or more) attribute values are related - For example, the 'age' and 'birth-date' attribute of a person. That is, 'age' can be determined from the current (today's) date and the value of that person's 'birth-date'. Hence the attribute 'age' is called a derived attribute and the attribute 'birth-date' is called as stored attribute.

Complex attributes: The composite and multivalued attributes can be nested by grouping components of a composite attribute between parentheses '()' and separating the components with commas, and by displaying multivalued attributes between braces ''. Such attributes are called complex attributes. For example, workon(project, hours)

Null value attributes: In some cases, a particular entity may not have an applicable values for an attribute. For example, 'apartment number' attribute of an address applies only to addresses that are in apartment building and not to other types of residences. For such situations, a special value called NULL is created.

Q9. What do you mean by role names in E-R model? How it is useful in E-R diagram?

Each entity type that participate in a relationship type plays a particular role in the relationship. The role names signifies the role that a participating entity from the entity type plays in each relationship instances, and helps to explain what the relationship means. For example, in a relationship between entity types EMPLOYEE and DEPARTMENT, EMPLOYEE plays the role of 'employee or worker' and DEPARTMENT play the role of 'department or employer'.

Role names are not necessary in relationship types where all the participating entity types are distinct since each participating entity type names can be used as the role name. However, in some cases the same entity type participates more than once in a relationship type in different roles. In such cases the role names becomes essential for distinguishing the meaning of each participation. Such relationship types are called recursive relationships. For an example, in the SUPERVISION relationship type shown below (see Figure 2) relates an employee to a supervisor, where both employee and supervisor entities are members of the same EMPLOYEE entity type. Hence, the EMPLOYEE entity type participate twice in SUPERVISION; once in the role of *supervisor*, and once in the role of *subordinate*.

Q10. Explain different constraints on relationships in E-R model

Constraints on relationships types are used to limit the possible combinations of entities that may participate In the corresponding relationships set. Two types of relationships constraints are cardinality ratio and participation.

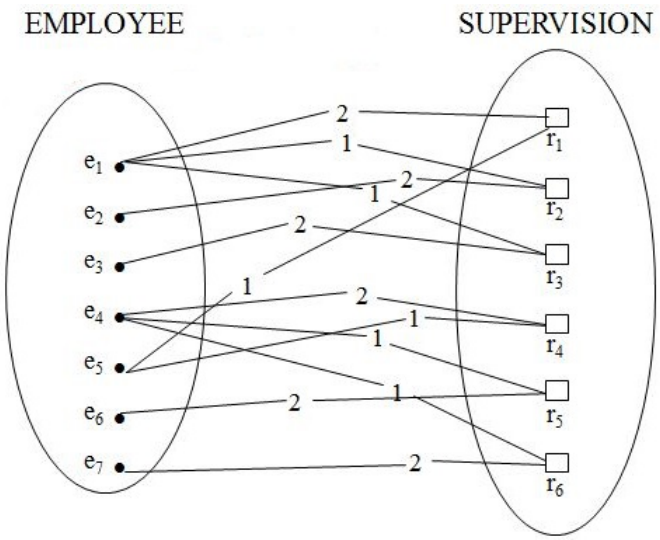


Figure 2: A recursive relationship SUPERVISION between EMPLOYEE in the *supervisor* role (1) and EMPLOYEE in the *subordinate* role (2)

Cardinality ratio: The cardinality ratio for a binary relationships specifies the maximum number of relationships instances that an entity can participate in. The possible cardinality ratios for binary relationship types are 1:1, 1:N, N:1 and M:N.

For example, a 1:1 binary relationship MANAGES (Figure 3) which relates a DEPARTMENT entity to the EMPLOYEE who manages that department. This represent a constraint - an employee can manage one department only and a department can have one manager only.

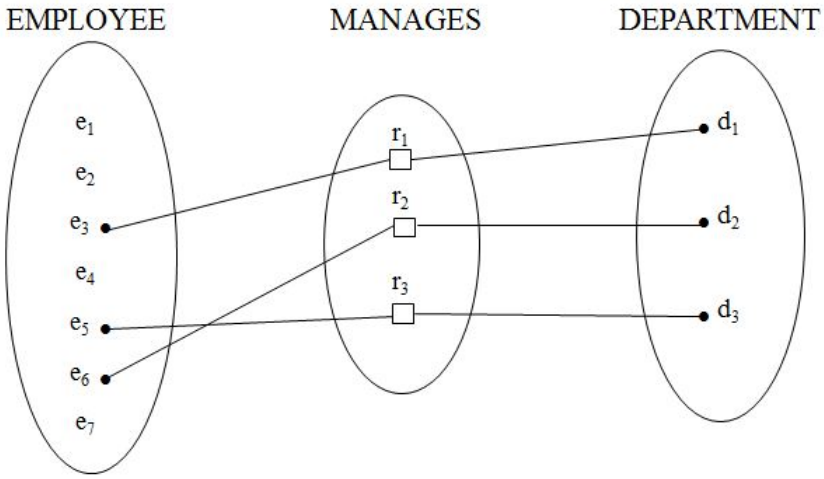


Figure 3: A 1:1 relationship MANAGES

[**Give other examples with diagram**]. Cardinality ratios for binary relationships are represented on E-R diagram by displaying 1, N and M on the diamonds.

Participation constraint: The participation constraint specifies whether the existence of an entity depends on it's being related to another entity via relationship types. These are two types participation constraints:total and partial.

If a company policy states that every employee must work for a department, then an employee entity exists only if it participate in at least one relationship instance. Therefor, the participation of EMPLOYEE entity type in a relationship type WORKS_FOR with DEPARTMENT entity type is called a *total* participation. However, the participation of EMPLOYEE entity type in MANAGES relationship type with DEPARTMENT entity type is *partial* meaning that some or part of the set of employee entities are related to some department entity via MANAGES.

In ER diagrams, total participation is displayed as a double line connecting the participating entity type to the relationship, where as partial participation is represented by a single line (See Figure 4.

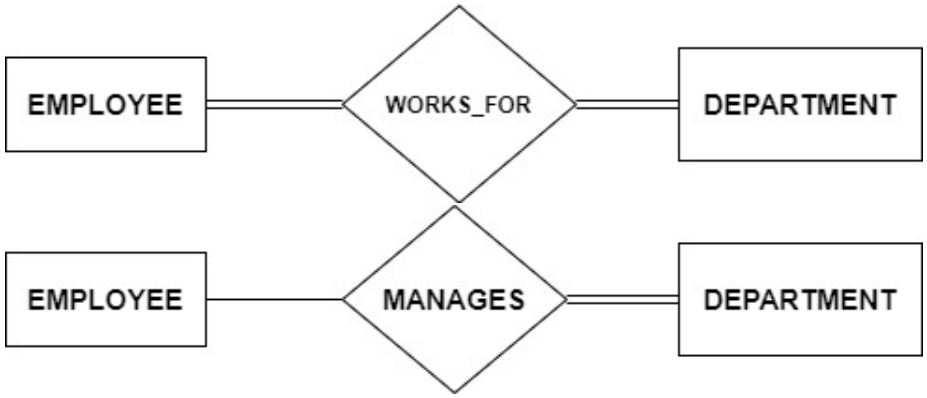


Figure 4: Representation of participation constraint

Q11. Explain weak entity type with example.

Entity type that do not have key attribute of their own are called weak entity types. Entities belonging to a weak entity type are identified by being related to specific entities from another entity type in combination with one of their attribute values. The other entity type is called as *identifying* or *owner entity type* and the relationship type that relates a weak entity type to its owner is called as *identifying relationship* of the weak entity type. A weak entity type always has a total participation constraint with respect to its identifying relationship because a weak entity cannot be identified without an owner entity.

A weak entity type normally has a partial key, which is the set of attributes that can uniquely identify weak entities that related to the same owner entity. For example an entity type DEPENDENT is a weak entity type and it is related to EMPLOYEE which is the owner entity type to keep track of the dependents of each employee. In ER diagram, both a weak entity type and its identifying relationship are distinguished by surrounding their boxes and diamonds with double lines (see Figure 5)

Figure 5: Weak Entity Type

Extended E-R Features

Q.12 Explain specialization and generalization in EER data model using suitable examples

Specialization

Specialization is the process of defining a set of sub-classes of an entity type; this entity type is called the super-class of the specialization. The set of sub-classes that form a specialization is defined on the basis of some distinguishing characteristics of the entities in the super-class. For example, the set of sub-classes SECRETARY, ENGINEER, TECHNICIAN is a specialization of the super-class EMPLOYEE that distinguish among employee entities based on the 'job type' of each employee entity.

The subclass that define a specialization are attached by lines to a circle[1] that represent the specialization, which is connected to the super-class. The subset symbol on each line connecting a subclass to the circle indicate the direction of the super-class/subclass relationship.

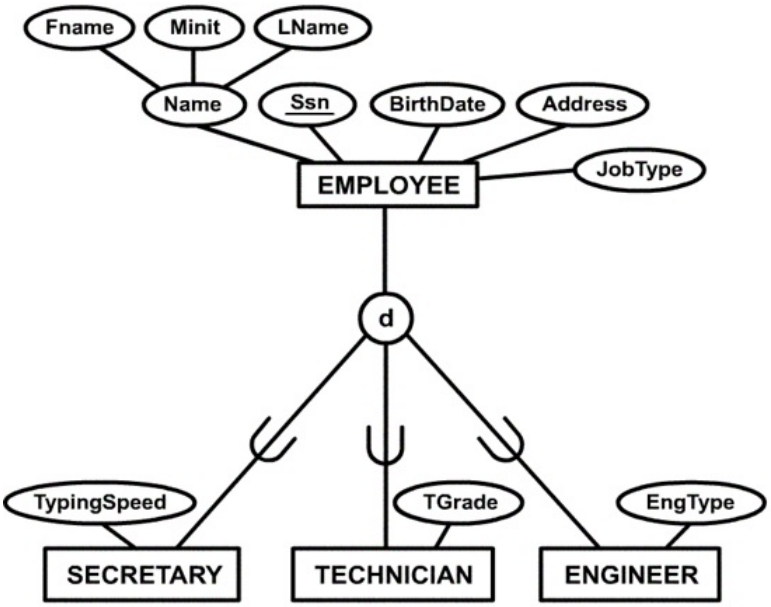


Figure 6: EER diagram to represent subclasses and specialization

Generalization

Generalization is a reverse process of abstraction in which we suppress the difference among several entity types, identify their common features and generalize them into a single super-class of which the original entity type are special sub-classes.

For example, consider the entity types CAR and TRUCK. Because they have several common attributes, they can be generalized into the entity type VEHICLE. Both CAR and TRUCK are now sub-classes of the generalized super-class VEHICLE.

[1] A specialization may also consist of a 'singe' subclass only. In such case, it is not necessary to use the circle notation.

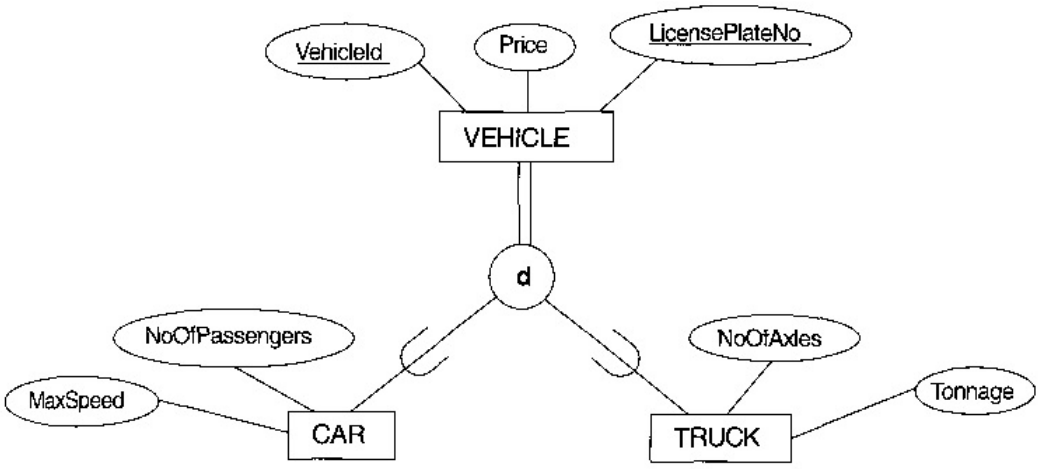


Figure 7: Generalization. (a) Two entity types, CAR and TRUCK. (b) Generalizing CAR and TRUCK into the superclass VEHICLE

Q.13 Explain the constraints on specialization and generalization

Specialization can determine exactly the entities that will become members of each subclass by planning a condition on the values of some attribute of the super-class. Such sub-classes are called predicate-defined (or condition-defined) sub-classes. For example, if the EMPLOYEE entity type has an attribute 'job-type', as shown in Figure 8, we can specify the condition of membership in the SECRETARY subclass by the condition (job-type='secretary'), which is the defining predicate of the subclass. In EER predicate-defined sub-class is represented by writing the predicate condition next to the line that connects the sub-class to the specialization circle.

If 'all' sub-classes in a specialization have their membership condition on the same attribute of the super-class, the specialization itself is called an attribute-defined specialization and the attribute is called the defining attribute of the specialization. Attribute-defined specialization is represented by placing the defining attribute name next to the arc from the circle to super-class, as shown in Figure 8.

Another constraints that may apply to specialization are disjointness constraint and completeness constraint.

Disjointness constraint specify that an entity can be a member of 'at most' one of the sub-classes of the specialization. A specialization that is attribute-defined implies the disjointness constraint if the attribute used to define the membership predicate is single-valued. If the subclass are not constrained to be disjoint, their sets of entities may overlap; that is, the same entity may be a member of more than one sub-class of the specialization. In EER diagram, 'd' in the circle stands for 'disjoint' and 'o' in the circle stands for overlap.

Completeness constraint may be total or partial. A total specialization constraint specifies that 'every' entity in the super-class must be a member of at least on sub-class in the specialization, if not it is known as partial specialization. Total participation in EER diagram is shown by using a double line to connect the super-class to the circle and a single line is used to display a partial specialization.

Both the disjointness and completeness constraints are independent. Hence in EER the following four possible constraints on specialization are possible:

1. disjoint, total

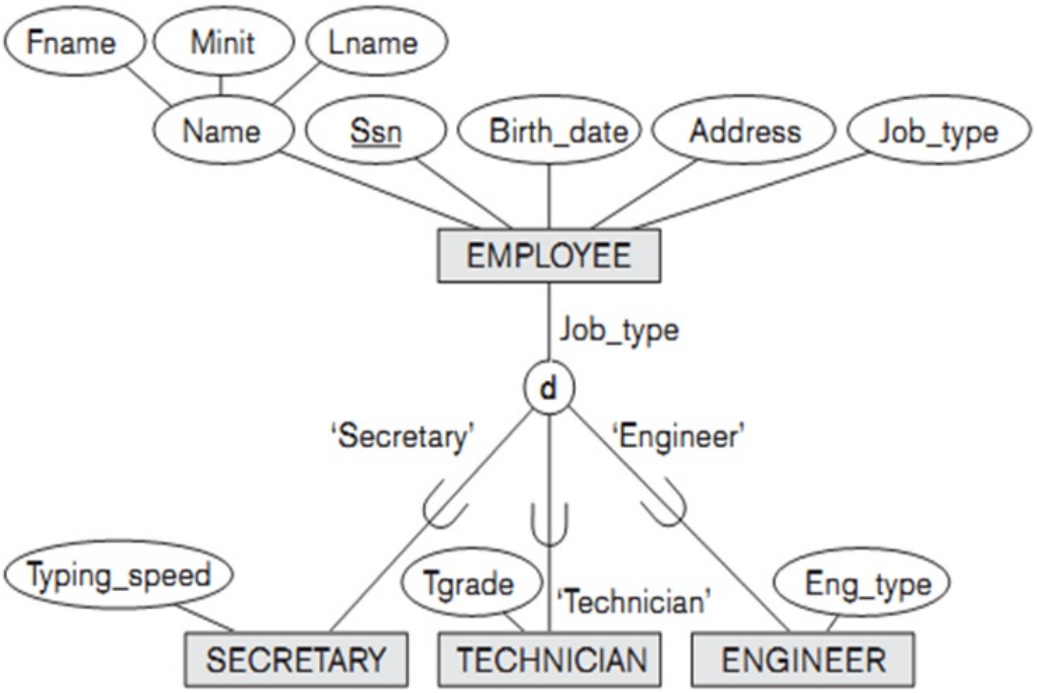


Figure 8: EER diagram notation for an attribute-defined specialization on JobType

2. disjoint, partial
3. overlap, total
4. overlap, partial

The Relational Model

Q14. Explain relation model concept

The relational model represents the database as a collection of *relations*. Informally, each relation resembles a table of values, or, to some extent, a *flat* file of records. When a relation is thought of as a **table** of values, each row in the table represents a collection of related data values.

In the formal relation model terminology, a row is called a *tuple*, a column header is called an *attribute*, and the table is called a *relation*. The data type describing the types of values that can appear in each column is represented by a *domain* of possible values.

A **relation schema** R, denoted by $R(A_1, A_2, ..., A_n)$ is made up of a relation name R and a list of attributes $A_1, A_2, ..., A_n$. Each **attribute** A_i is the name of a role played by some domain D in the relation schema R. D is called the **domain** of A_i and is denoted by $dom(A_i)$. A relation schema is used to describe a relation; R is called the name of this relation. The **degree** (or **arity**) of a relation is the number of attributes n of its relation schema. Figure 9 shows a relational database schema for COMPANY database.

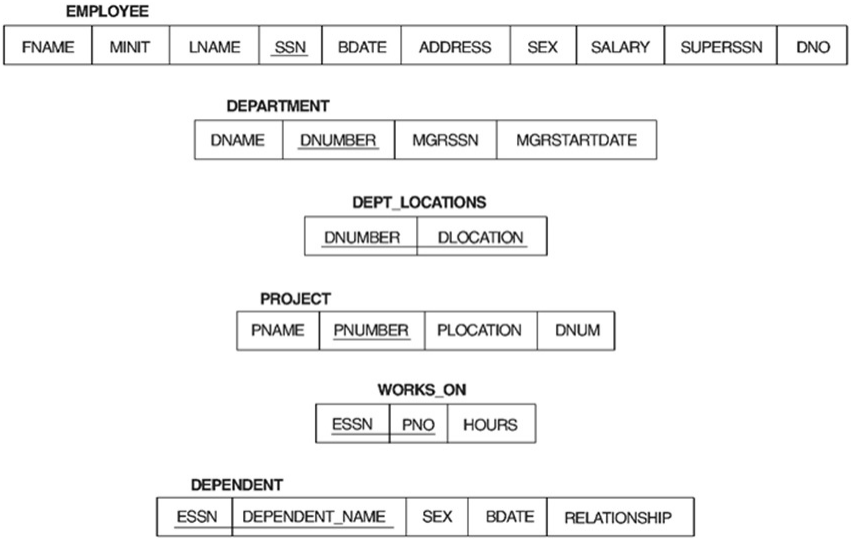


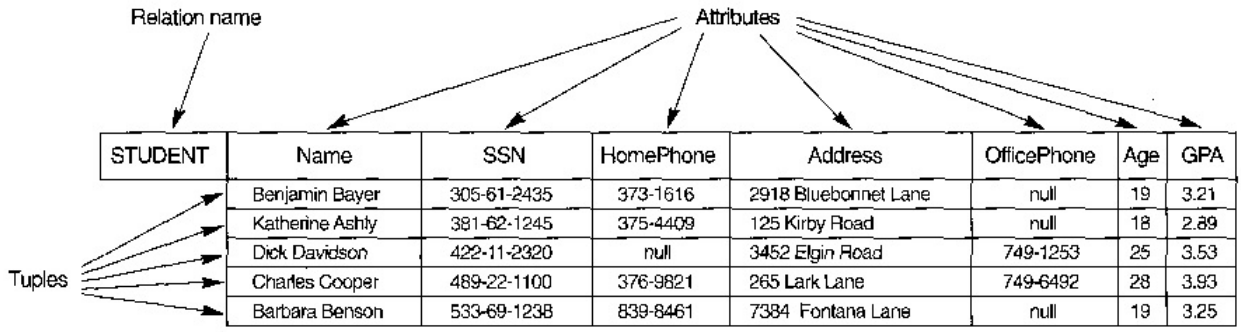
Figure 9: Schema diagram for the COMPANY relational database schema

A **relation** (or **relation state**) r of the relation schema $R(A_1, A_2, ..., A_n)$, also denoted by $r(R)$, is a set of tuples $r = \{t_1, t_2, ..., t_m\}$. Each tuple t is an ordered list of n values $t =< v_1, v_2, ...,_v n >$, where each value $v_i, 1 \leq i \leq n$, is an element of $dom(A_i)$ or is a special NULL value. The ith value in tuple t, which corresponds to the attribute A_i, is referred to as $t[A_i]$.

Figure 10 shows an example of a STUDENT relation, which corresponds to the STUDENT schema just specified. Each tuple in the relation represents a particular student entity.

From the earlier definition, a relation (or relation state) $r(R)$ is a mathematical relation of degree n on the domains $dom(A_1), dom(A_2), ..., dom(A_n)$, which is a subset of the Cartesian product of the domain that define R:

$r(R) \subseteq (dom(A_1) \times dom(A_2) \times .... \times dom(A_n))$



STUDENT	Name	SSN	HomePhone	Address	OfficePhone	Age	GPA
	Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	null	19	3.21
	Katherine Ashly	381-62-1245	375-4409	125 Kirby Road	null	18	2.89
	Dick Davidson	422-11-2320	null	3452 Elgin Road	749-1253	25	3.53
	Charles Cooper	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
	Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	null	19	3.25

Figure 10: Schema diagram for the COMPANY relational database schema

Q15. Explain relation model constraints

The schema-based constraints in the relational model include, domain constraints, key constraints, constraints on NULL, entity integrity constraints and referential integrity constraints.

Domain constraints

Domain constraints specify that within each tuple, the value of each attribute must be an atomic value from its domain. For example, a date does not have a 'real' type value.

Key constraints

A relation is defined as a set of tuples. By definition, all elements of a set are distinct; hence all tuple in a relation must also be distinct. This means that no two tuples can have the same combination of values for all their attributes.

A **superkey** is a set of one or more attributes that, taken collectively, allows to identify uniquely an entity in the entity set. For example, the 'ssn' attribute of the 'employee' relation is sufficient to distinguish on employee tuple from another.

A superkey may contain redundant attributes. That is if 'ssn' is a superkey of the 'employee' relation, then 'ename' and 'ssn' is also a superkey. In relational model, it is often important in superkeys for which no proper subset is a superkey. Such minimal superkey are called **candidate key**. Thus, a candidate key is a minimum distinct sets of attribute(s) that can uniquely identify a tuple in a relation.

It is common that, a relation schema may have more one candidate keys. Therefore, one of the candidate keys should be designate as the principal means of identifying tuples within a relation by a database designer. Such a candidate key is known as **primary key**. Then the remaining candidate keys are called **alternate keys**.

Constraints on NULL

An important concept is that of NULL values, which are used to represent the values of attributes that may be unknown or may not apply to a tuple. A special value called **NULL** is used in these cases. Therefore, constraints on NULL specifies whether NULL values are or are not permitted. For example, if every 'student' tuple must have a valid, non-NULL value for the 'name' attribute, then the 'name' attribute of the 'student' relation is considered to be 'NOT NULL'.

Entity integrity constraint

The entity integrity constraint stats that no primary key value can be NULL. This is because the primary key value is used to identify individual tuples in a relation. Having NULL value for the primary key implies that it cannot identify some tuples. For example, if two or more tuples had NULL for their primary key, we might not be able to distinguish them if we tried to referenced them from other relations.

Referential integrity constraint

The referential integrity constraint is specified between two relations and is used to maintain the consistency among tuples in the two relations. Informally, the referential integrity constraint stats that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation.

The referential integrity constraint is closely relates with foreign key. A set of attribute in a relation schema R_1, is said to be **foreign key** of R_1, that references relation R_2, if and only if it satisfies the following rules:

1. The foreign key attribute of R_1 should have the same domain as the primary key attribute of R_2; the foreign key attribute are said to reference or refer to the relation R_2.

2. A value of foreign key in a tuple t_1 of the current state $r_1(R_1)$ either occurs as a value of primary key for some tuple t_2 in the current state $r_2(R_2)$ or is NULL.

We can diagrammatically display referential integrity constraints by drawing a directed arc from each foreign key to the relation it references. Figure 11 shows the COMPANY schema with the referential integrity constraints displayed in this manner.

Q16. Explain different relational algebra operations.

The relational algebra is often considered to be an integral part of the relational model. Its operations can be divided into two groups. One group include set operations from mathematical set theory[2]; these are applicable because each relation is defined to be a set of tuples in the formal relational model. The other group operations developed specifically for relational database[3]. Another interesting classification is the set of fundamental or complete set of relation algebra operations[4]; that is, any of the other original relational algebra operations can be expressed as a 'sequence of operations from this set'.

The SELECT operation

SELECT operation is an unary operation in the sense that this operate on single relation. The SELECT operation, denoted by (σ) and pronounced as 'sigma', $\sigma_F(R)$, is a set of tuples 't' in 'R' such that 't' must satisfy the condition defined in the formula 'F', where 'F' involves:

1. operand that are constants or attribute names

2. the arithmetic comparison operators $<, =, >, \neq, \leq \quad and \quad \geq$

[2]Set operations include UNION, INTERSECTION, SET DIFFERENCE and CARTESIAN PRODUCT

[3]SELECT, PROJECT, RENAME and JOINS are the operations developed specifically for relational database

[4]SELECT, PROJECT, UNION, SET DIFFERENCE and CARTESIAN PRODUCT are considered to be fundamental/complete set

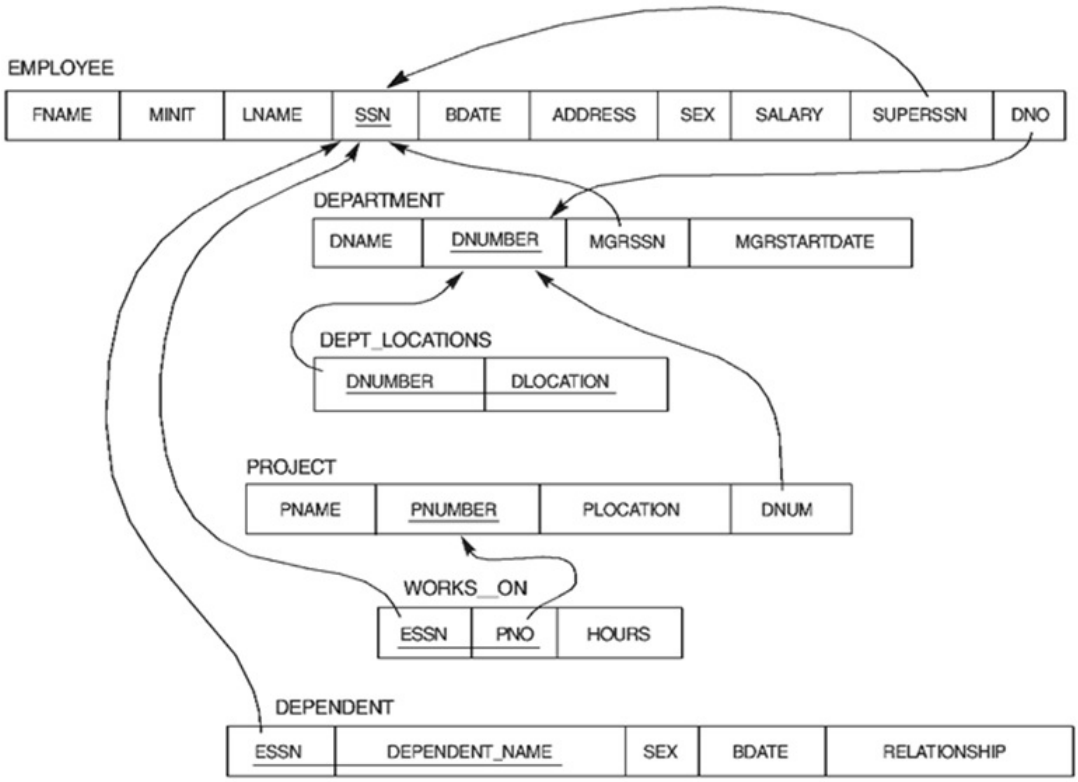


Figure 11: Referential integrity constraints displayed on the COMPANY relational database schema

3. the logical operators $\wedge(and), \vee(or)$ and $\neg(not)$

Therefore the SELECT operation can be viewed as a filter that keeps only those tuples that satisfy a qualifying condition.

The components in the formula 'F' are organized as the clauses of the form:

<attribute name><comparison operator><constant value>

or

<attribute name><comparison operator><attribute name>

clauses can be arbitrarily connected by the logical operators.

For example, to select the 'employee' tuples who are working in the department number 4 is represented as

$$\sigma_{dno=4}(EMPLOYEE)$$

The PROJECT operation

The PROJECT operation is used to select certain columns/attributes from the table and discards the other columns/attributes. The idea behind this operation is that, it takes a relation 'R', remove and/or rearrange some of the attributes. The general form of the PROJECT operation is:

$$\Pi_{<attribute_list>}(R)$$

where Π (pi) is the symbol used to represent the PROJECT operation, and <attribute_list>is the desired list of attributes of the relation 'R'.

For example, to list each employee's name and salary, the following PROJECT operation can be used:

$$\Pi_{name,salary}(EMPLOYEE)$$

The result of the PROJECT operation has only the attributes specified in <attribute_list>in the same order as they appear in the list. Hence, its degree is equal to the number of attributes in <attribute_list>. If the attribute list include only nonkey attributes of 'R', duplicate tuples are likely to occur. The project operation removes any duplicate tuples. So the result of the PROJECT operation is a set of tuples, and hence a valid relation. The number of tuples in a relation resulting from a PROJECT operation is always less than or equal to the number of tuples in 'R'. If the projection list is a superkey of 'R' - that is, it includes some key of 'R' - the resulting relation has the same number of tuples of 'R'.

The RENAME operation

It is possible to write the operation as a single relational algebra expression by nesting the operations, or we can apply one operation at a time and create intermediate result relation. In later case, we must give names to the relation that hold the intermediate results.

For example, the relational algebra to retrieve the name and salary of all employees who work in department number 5 is:

$$\Pi_{name,salary}(\sigma_{dno=5}(EMPLOYEE)$$

Alternatively, the above operation can break up by giving a name to each intermediate relation:

$$DEP5\_EMPS \longleftarrow \sigma_{dno=5}(EMPLOYEE)$$
$$RESULT \longleftarrow \Pi_{name,salary}(DEP5\_EMPS)$$

The RENAME operation is also used to rename either the relation name or the attribute names or both - as a unary operator. The general RENAME operation when applied to a relation 'R' of degree 'n' is denoted by any of the following three forms.

$$\rho_{S(B_1,B_2,.....,B_n)}(R) \text{ or } \rho_S(R) \text{ or } \rho_{(B_1,B_2,....,B_n)}(R)$$

where the symbol ρ (rho) is used to denote the RENAME operator, S is the new relation name and $B_1, B_2, ..., B_n$ are the new attribute name. The first expression renames both the relation and its attributes, the second renames the relation only and the third renames the attribute only.

For example, to rename some attribute of the DEPARTMENT relation, the following RENAME operation can be used:

$$\rho_{dnum,dname,location}(DEPARTMENT)$$

The UNION, INTERSECTION and SET-DIFFERENCE operations

Set theoretic operations are used to merge the elements of two sets in various ways including UNION, INTERSECTION and SET DIFFERENCE (also called MINUS). These are binary operations; that is, each is applied to two sets (of tuples). In order to apply these operations in relational database, the two relations on which any of these operations are applied must satisfy the *union compatibility* condition.

Two relations $R(A_1, A_2, ..., A_n)$ and $S(B_1, B_2, ..., B_n)$ are said to be **union compatible** if they have the same degree 'n' and $dom(A_i) = dom(B_i)$ for $1 \leq i \leq n$. This means hat the two relations have the same number of attributes and each corresponding pair of attributes has the same domain.

The three operations can be defined as follows:

UNION: The result of this operation, denoted by $R \cup S$, is a relation that includes all tuples that are either in 'R' or in 'S' or in both 'R' and 'S'. Duplicate tuples are eliminated.

INTERSECTION: The result of this operation, denoted by $R \cap S$, is a relation that includes all tuples that are in both 'R' and 'S'.

SET DIFFERENCE: The result of this operation, denoted by $R - S$, is a relation that include all tuples that are in 'R' but not in 'S'.

The CARTESIAN PRODUCT (CROSS PRODUCT) operation

Cartesian product is a binary set operation, but the relation on which it is applied do not have to be union compatible. If 'R' and 'S' are the two relations, the resulting relation, say 'Q', after the Cartesian product will have one tuple for each combination of tuples - one from 'R' and one from 'S'. Hence if 'R' has n_R tuples and 'S' has n_S tuples, then $R \times S$ will have $n_R * n_S$ tuples.

Give an example

The JOIN operation

The join operation, denoted by $\bowtie$, is used to combine related tuples from two relations into single tuples. This operation is very important for any relational database with more than a single relation because it allows to process relationships among relations.

The result of the JOIN is a relation 'Q' with $n+m$ attributes $Q(A_1, A_2, ..., A_n, B_1, B_2, ..., B_m)$ in that order. 'Q' has one tuple for each combination of tuple - one from 'R' and one from 'S' - whenever the combination satisfies the join condition. The join condition is specified on attributes from the two relations 'R' and 'S' and is evaluated for each combination of tuples. Each tuple combination for which the join condition evaluates to TRUE is included in the resulting relation 'Q' as a single combined tuple.

The general join condition is of the form

<condition>AND <condition>AND .....AND <condition>

where each condition is of the form $A_i \theta B_j$, where A_i is an attribute of 'R', B_j is an attribute of 'S', A_i and B_j have the same domain and θ (theta) is one of the comparison operator $\{<, =, > , \neq, \leq \ and \ \geq\}$. A join operation with such general join condition is called a **THETA JOIN**. Tuples whose join attributes are NULL or for which the join condition is FALSE do not appear in the result.

Give an example

Variations of JOIN: The EQUIJOIN and NATURAL JOIN

The most common use of JOIN involves join condition with equality comparison only. Such a join, where the only comparison operator used is '=', is called an EQUIJOIN.

The result of EQUIJOIN will have one or more pair of attributes that have identical values in every tuple. Because of these superfluous values, a new operator called NATURAL JOIN - denoted by '*' - was created to get rid of the second (superfluous) attribute in an EQUIJOIN condition. The definition of NATURAL JOIN requires that the two join attributes (or each pair of join attributes) have the same name in both relations. If this is not the case, a RENAME operation is applied first.

Q17. A problem with relation algebra

Given the relation[5]:

DEPARTMENT(dno, dname, location)
EMPLOYEE(ssn, ename, address, dno)
PROJECT(pno, pname, dno)

[5]This is only a sample question

WORK_ON(essn, pno, hours)
write relational algebra queries for the following:

1. Retrieve the name and address of all employees who work for the 'Research' department.
2. Make a list of all project numbers for projects that involve an employee whose name is 'Smith'.