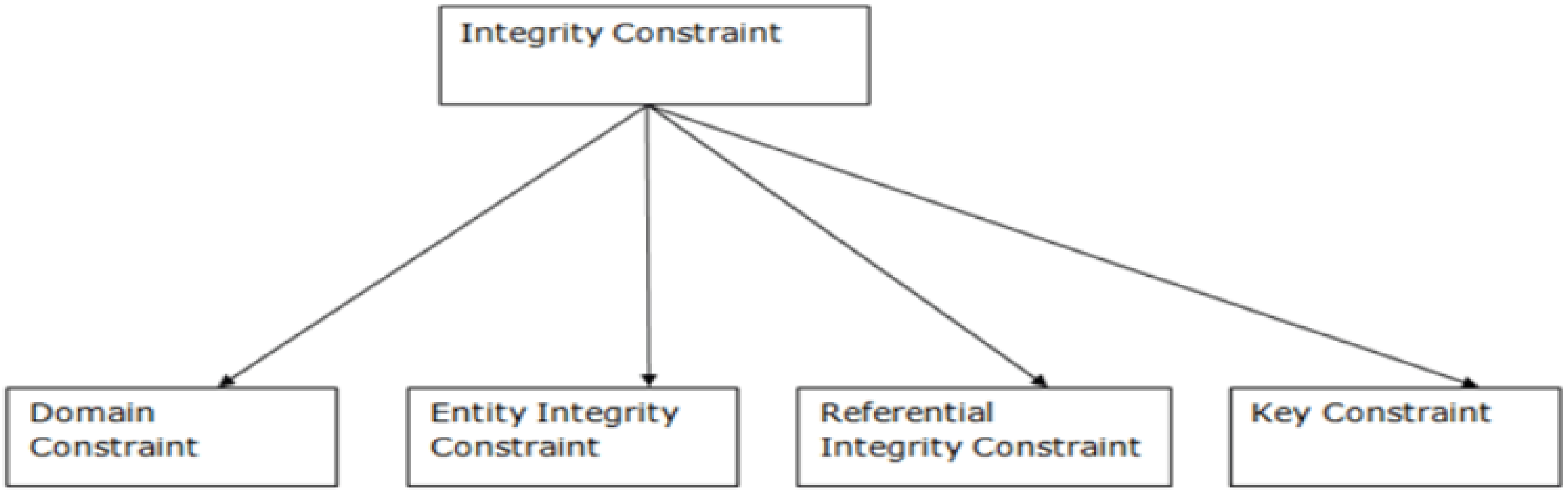
Q.1 What is integrity constraint? Write the various constraint avaible, in brief.

Integrity Constraints

← Prev Next →

- Integrity constraints are a set of rules. It is used to maintain the quality of information.
- Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.
- Thus, integrity constraint is used to guard against accidental damage to the database.

Types of Integrity Constraint



1. Domain constraints

- Domain constraints can be defined as the definition of a valid set of values for an attribute.
- The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain.

Example:

ID	NAME	SEMENSTER	AGE
1000	Tom	1^{st}	17
1001	Johnson	2^{nd}	24
1002	Leonardo	5^{th}	21
1003	Kate	3^{rd}	19
1004	Morgan	8^{th}	A

Not allowed. Because AGE is an integer attribute

2. Entity integrity constraints

- The entity integrity constraint states that primary key value can't be null.
- This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
- A table can contain a null value other than the primary key field.

Example:

EMPLOYEE

EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

Not allowed as primary key can't contain a NULL value

3. Referential Integrity Constraints

- A referential integrity constraint is specified between two tables.
- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.

Example:

EMPLOYEE

EMP_ID	EMP_NAME	SALARY
123	Jack	30000
142	Harry	60000
164	John	20000
	Jackson	27000

Not allowed as primary key can't contain a NULL value

3. Referential Integrity Constraints

- A referential integrity constraint is specified between two tables.
- In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.

$$\Pi_{\text{Name, Department}}(\sigma_{\text{salary>25000}}(\text{EMPLOYEE}))$$

Q.2 Why we use key in DBMS and define. Various key avaible in DBMS system.

- A Key is an attribute or a set of attributes in a relation that identifies a tuple (record) in a relation.
- The keys are defined in a table to access or sequence the stored data quickly and smoothly.
- They are also used to create relationship between different tables.

Types of Keys in Database

1. Primary Key
2. Candidate Key
3. Alternate Key
4. Super Key
5. Composite Key
6. Foreign Key
7. Unique Key

Primary Key

- Which is ***Unique & Can't be have NULL Value***
- Is the column you choose to maintain uniqueness in a table at row level.
- Here in ***Employee*** table we can choose either ***EmployeeID*** or ***SSN*** column for a PK.
- EmployeeID is preferable choice because SSN is a secure (PII) value.
- Primary key is the minimal super keys. In the ER diagram primary key is represented by underlining the primary key attribute.
- Ideally a primary key is composed of only a single attribute.
- But it is possible to have a primary key composed of more than one attribute

Employee
EmployeeID
EmployeeName
SSN
DeptID
DOB

To define a field as primary key, following conditions had to be met :

- No two rows can have the same primary key value.
- Every row must have a primary key value.
- The primary key field cannot be null.
- Value in a primary key column can never be modified or updated, if any foreign key refers to that primary key

Candidate Key

- Are individual columns in a table that qualifies for uniqueness of each row/tuple.
- Here in ***Employee*** table ***EmployeeID*** & ***SSN*** are eligible for a ***Primary Key*** and thus are ***Candidate keys***.
- Candidate Keys are super keys for which no proper subset is a super key. ***In other words candidate keys are minimal super keys.***

Employee
EmployeeID
EmployeeName
SSN
DeptID
DOB

Alternate Key

- Candidate column other the Primary column, like if ***EmployeeID*** is set for a PK then ***SSN*** would be the Alternate key.

Employee
EmployeeID
EmployeeName
SSN
DeptID
DOB

Super Key

- If you add any other Column / Attribute to a Primary Key then it become a Super Key, like ***EmployeeID* + *EmployeeName*** is a Super Key.
- Super key stands for superset of a key.
- ***A Super Key is a set of one or more attributes that are taken collectively and can identify all other attributes uniquely.***

Employee
EmployeeID
EmployeeName
SSN
DeptID
DOB

Composite Key

- If a table do have a single column that qualifies for a Candidate key, then you have to select 2 or more columns to make a row unique.
- Like if there is no EmployeeID or SSN columns, then you can make ***EmployeeName + DateOfBirth*** (DOB) as ***Composite Primary Key***. But still there can be a narrow chance of duplicate rows.

Employee
EmployeeID
EmployeeName
SSN
DeptID
DOB

Foreign Key

- Here in below tables ***DeptID*** of ***Department*** table is Primary Key where as ***DeptID*** of ***Employee*** is an ***Foreign key***.
- It means it has referred to another table. This concept is also know as ***Referential Integrity***.

Employee
EmployeeID
EmployeeName
SSN
DeptID
DOB

Department
DeptID
DeptName

Unique Key

- ***Unique key*** is same as primary with the difference being the existence of null.

- Unique key field allows one value as NULL value.

Employee
EmployeeID
EmployeeName
SSN
EmailID
DOB

Q.3 Define Relational algebra with their various operation.

- Relational algebra is the basic set of operations for the relational model
- These operations enable a user to specify **basic retrieval requests** (or **queries**)
- The result of an operation is a *new relation*, which may have been formed from one or more *input* relations
  - This property makes the algebra “closed” (all objects in relational algebra are relations)
- The **algebra operations** thus produce new relations
  - These can be further manipulated using operations of the same algebra
- A sequence of relational algebra operations forms a **relational algebra expression**
  - The result of a relational algebra expression is also a relation that represents the result of a database query (or retrieval request)

- Relational Algebra consists of several groups of operations
  - Unary Relational Operations
    - SELECT (symbol: σ (sigma))
    - PROJECT (symbol: π (pi))
    - RENAME (symbol: ρ (rho))
  - Relational Algebra Operations From Set Theory
    - UNION ($\cup$), INTERSECTION ($\cap$), DIFFERENCE (or MINUS, –)
    - CARTESIAN PRODUCT (**x**)
  - Binary Relational Operations
    - JOIN (several variations of JOIN exist)
    - DIVISION
  - Additional Relational Operations
    - OUTER JOINS, OUTER UNION
    - AGGREGATE FUNCTIONS (These compute summary of information: for example, SUM, COUNT, AVG, MIN, MAX)

- The SELECT operation (denoted by σ (sigma)) is used to select a *subset* of the tuples from a relation based on a **selection condition**.
  - The selection condition acts as a **filter**
  - Keeps only those tuples that satisfy the qualifying condition
  - Tuples satisfying the condition are *selected* whereas the other tuples are discarded (*filtered out*)
- Examples:
  - Select the EMPLOYEE tuples whose department number is 4:

$$\sigma_{DNO = 4}\ (\text{EMPLOYEE})$$

  - Select the employee tuples whose salary is greater than $30,000:

$$\sigma_{SALARY > 30{,}000}\ (\text{EMPLOYEE})$$

- In general, the *select* operation is denoted by $\sigma_{\text{<selection condition>}}(R)$ where
  - the symbol σ (sigma) is used to denote the *select* operator
  - the selection condition is a Boolean (conditional) expression specified on the attributes of relation R
  - tuples that make the condition **true** are selected
    - appear in the result of the operation

tuples that make the condition **false** are filtered out discarded from the result of the operation

- PROJECT Operation is denoted by π (pi)
- This operation keeps certain *columns* (attributes) from a relation and discards the other columns.
  - PROJECT creates a vertical partitioning
    - The list of specified columns (attributes) is kept in each tuple
    - The other attributes in each tuple are discarded
- Example: To list each employee's first and last name and salary, the following is used:

$$\pi_{\text{LNAME, FNAME,SALARY}}(\text{EMPLOYEE})$$

- The general form of the *project* operation is:

$$\pi_{\text{<attribute list>}}(R)$$

  - π (pi) is the symbol used to represent the *project* operation
  - <attribute list> is the desired list of attributes from relation R.
- The project operation *removes any duplicate tuples*
  - This is because the result of the *project* operation must be a *set of tuples*
    - Mathematical sets *do not allow* duplicate elements.

- PROJECT Operation Properties
  - The number of tuples in the result of projection $\pi_{\text{<list>}}(R)$ is always less or equal to the number of tuples in R
    - If the list of attributes includes a *key* of R, then the number of tuples in the result of PROJECT is *equal* to the number of tuples in R
  - PROJECT is *not* commutative
    - $\pi_{\text{<list1>}} (\pi_{\text{<list2>}} (R)) = \pi_{\text{<list1>}} (R)$ as long as <list2> contains the attributes in <list1>

- The RENAME operator is denoted by ρ (rho)
- In some cases, we may want to *rename* the attributes of a relation or the relation name or both
  - Useful when a query requires multiple operations
  - Necessary in some cases (see JOIN operation later)
- The general RENAME operation ρ can be expressed by any of the following forms:
  - $\rho_{S\ (B1,\ B2,\ \ldots,\ Bn\ )}(R)$ changes both:
    - the relation name to S, *and*
    - the column (attribute) names to B1, B1, …..Bn
  - $\rho_S(R)$ changes:
    - the *relation name* only to S
  - $\rho_{(B1,\ B2,\ \ldots,\ Bn\ )}(R)$ changes:
    - the *column (attribute) names* only to B1, B1, …..Bn

- For convenience, we also use a *shorthand* for renaming attributes in an intermediate relation:
  - If we write:
    - RESULT $\leftarrow \pi_{\text{FNAME, LNAME, SALARY}}$ (DEP5_EMPS)
    - RESULT will have the *same attribute names* as DEP5_EMPS (same attributes as EMPLOYEE)
  - If we write:
    - RESULT (F, M, L, S, B, A, SX, SAL, SU, DNO)$\leftarrow$ $\pi_{\text{FNAME, LNAME, SALARY}}$ (DEP5_EMPS)
    - The 10 attributes of DEP5_EMPS are *renamed* to F, M, L, S, B, A, SX, SAL, SU, DNO, respectively

- UNION Operation
  - Binary operation, denoted by $\cup$
  - The result of 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
  - The two operand relations R and S must be "type compatible" (or UNION compatible)
    - R and S must have same number of attributes
    - Each pair of corresponding attributes must be type compatible (have same or compatible domains)

- Example:
  - To retrieve the social security numbers of all employees who either *work in department 5* (RESULT1 below) or *directly supervise an employee who works in department 5* (RESULT2 below)
  - We can use the UNION operation as follows:

$$\text{DEP5\_EMPS} \leftarrow \sigma_{\text{DNO}=5}\ (\text{EMPLOYEE})$$

$$\text{RESULT1} \leftarrow \pi_{\text{SSN}}(\text{DEP5\_EMPS})$$

$$\text{RESULT2(SSN)} \leftarrow \pi_{\text{SUPERSSN}}(\text{DEP5\_EMPS})$$

$$\text{RESULT} \leftarrow \text{RESULT1} \cup \text{RESULT2}$$

  - The union operation produces the tuples that are in either RESULT1 or RESULT2 or both

- INTERSECTION is denoted by $\cap$
- The result of the operation $R \cap S$, is a relation that includes all tuples that are in both R and S
  - The attribute names in the result will be the same as the attribute names in R
- The two operand relations R and S must be "type compatible"
- SET DIFFERENCE (also called MINUS or EXCEPT) is denoted by –
- The result of $R – S$, is a relation that includes all tuples that are in R but not in S
  - The attribute names in the result will be the same as the attribute names in R
- The two operand relations R and S must be "type compatible"

- Notice that both union and intersection are *commutative* operations; that is
  - $R \cup S = S \cup R$, and $R \cap S = S \cap R$
- Both union and intersection can be treated as n-ary operations applicable to any number of relations as both are *associative* operations; that is
  - $R \cup (S \cup T) = (R \cup S) \cup T$
  - $(R \cap S) \cap T = R \cap (S \cap T)$
- The minus operation is not commutative; that is, in general
  - $R - S \neq S - R$

- CARTESIAN (or CROSS) PRODUCT Operation
  - This operation is used to combine tuples from two relations in a combinatorial fashion.
  - Denoted by R(A1, A2, . . ., An) x S(B1, B2, . . ., Bm)
  - Result is a relation Q with degree n + m attributes:
    - Q(A1, A2, . . ., An, B1, B2, . . ., Bm), in that order.
  - The resulting relation state has one tuple for each combination of tuples—one from R and one from S.
  - Hence, if R has n_R tuples (denoted as $|R| = n_R$), and S has n_S tuples, then R x S will have $n_R * n_S$ tuples.
  - The two operands do NOT have to be "type compatible”

Q.4 Write the Relational algrbraic expression for the following questions :-
(Employee table given in Question)

- Display the employee name, department who has salary greater than 25000.
- Retrieves the social security numbers. Of all employees who either work in department no. 5 or directly supervise an employee who works in department no. 5
- Retrieve the names of female employees and their dependents.

Single expression versus sequence of relational operations (Example)

- To retrieve the first name, last name, and salary of all employees who work in department number 5, we must apply a select and a project operation
- We can write a *single relational algebra expression* as follows:
  - $\pi_{\text{FNAME, LNAME, SALARY}}(\sigma_{\text{DNO=5}}(\text{EMPLOYEE}))$
- OR We can explicitly show the *sequence of operations*, giving a name to each intermediate relation:
  - $\text{DEP5\_EMPS} \leftarrow \sigma_{\text{DNO=5}}(\text{EMPLOYEE})$
  - $\text{RESULT} \leftarrow \pi_{\text{FNAME, LNAME, SALARY}}(\text{DEP5\_EMPS})$

Relational Algebra Operations from Set Theory: UNION

- Example:
  - To retrieve the social security numbers of all employees who either *work in department 5* (RESULT1 below) or *directly supervise an employee who works in department 5* (RESULT2 below)
  - We can use the UNION operation as follows:

$$\text{DEP5\_EMPS} \leftarrow \sigma_{\text{DNO=5}} (\text{EMPLOYEE})$$

$$\text{RESULT1} \leftarrow \pi_{\text{SSN}}(\text{DEP5\_EMPS})$$

$$\text{RESULT2(SSN)} \leftarrow \pi_{\text{SUPERSSN}}(\text{DEP5\_EMPS})$$

$$\text{RESULT} \leftarrow \text{RESULT1} \cup \text{RESULT2}$$

  - The union operation produces the tuples that are in either RESULT1 or RESULT2 or both

Example of the result of a UNION operation

- UNION Example

Figure 6.3
Result of the UNION operation RESULT ← RESULT1 ∪ RESULT2.

RESULT1

Ssn
123456789
333445555
666884444
453453453

RESULT2

Ssn
333445555
888665555

RESULT

Ssn
123456789
333445555
666884444
453453453
888665555

Relational Algebra Operations from Set Theory: CARTESIAN PRODUCT (cont.)

- To keep only combinations where the DEPENDENT is related to the EMPLOYEE, we add a SELECT operation as follows
- Example (meaningful):
  - FEMALE_EMPS $\leftarrow \sigma_{SEX='F'}$(EMPLOYEE)
  - EMPNAMES $\leftarrow \pi_{FNAME, LNAME, SSN}$ (FEMALE_EMPS)
  - EMP_DEPENDENTS $\leftarrow$ EMPNAMES x DEPENDENT
  - ACTUAL_DEPS $\leftarrow \sigma_{SSN=ESSN}$(EMP_DEPENDENTS)
  - RESULT $\leftarrow \pi_{FNAME, LNAME, DEPENDENT\_NAME}$ (ACTUAL_DEPS)
- RESULT will now contain the name of female employees and their dependents

Q.5 what are the various aggregate functions used/available in DBMS? Explain with an example.

Sum, average, maximum, minimum, count with example

Additional Relational Operations: Aggregate Functions and Grouping

- A type of request that cannot be expressed in the basic relational algebra is to specify mathematical **aggregate functions** on collections of values from the database.
- Examples of such functions include retrieving the average or total salary of all employees or the total number of employee tuples.
  - These functions are used in simple statistical queries that summarize information from the database tuples.
- Common functions applied to collections of numeric values include
  - SUM, AVERAGE, MAXIMUM, and MINIMUM.
- The COUNT function is used for counting tuples or values.

Aggregate Function Operation

- Use of the Aggregate Functional operation $\mathcal{F}$
  - $\mathcal{F}_{\text{MAX Salary}}$ (EMPLOYEE) retrieves the maximum salary value from the EMPLOYEE relation
  - $\mathcal{F}_{\text{MIN Salary}}$ (EMPLOYEE) retrieves the minimum Salary value from the EMPLOYEE relation
  - $\mathcal{F}_{\text{SUM Salary}}$ (EMPLOYEE) retrieves the sum of the Salary from the EMPLOYEE relation
  - $\mathcal{F}_{\text{COUNT SSN, AVERAGE Salary}}$ (EMPLOYEE) computes the count (number) of employees and their average salary
    - Note: count just counts the number of rows, without removing duplicates

Q.6 why we use join operation in DBMS. Write the various join operations with an example.

Binary Relational Operations: JOIN

- JOIN Operation (denoted by)
  - The sequence of CARTESIAN PRODECT followed by SELECT is used quite commonly to identify and select related tuples from two relations
  - A special operation, called JOIN combines this sequence into a single operation
  - This operation is very important for any relational database with more than a single relation, because it allows us *combine related tuples* from various relations
  - The general form of a join operation on two relations R(A1, A2, . . ., An) and S(B1, B2, . . ., Bm) is:

$$R \quad {}_{\text{<join condition>}} S$$

  - where R and S can be any relations that result from general *relational algebra expressions*.

Binary Relational Operations: JOIN (cont.)

- Example: Suppose that we want to retrieve the name of the manager of each department.
  - To get the manager's name, we need to combine each DEPARTMENT tuple with the EMPLOYEE tuple whose SSN value matches the MGRSSN value in the department tuple.
  - We do this by using the join operation.
  - DEPT_MGR $\leftarrow$ DEPARTMENT $_{\text{MGRSSN=SSN}}$ EMPLOYEE
- MGRSSN=SSN is the join condition
  - Combines each department record with the employee who manages the department
  - The join condition can also be specified as DEPARTMENT.MGRSSN= EMPLOYEE.SSN

Example of applying the JOIN operation

DEPT_MGR

Dname	Dnumber	Mgr_ssn	...	Fname	Minit	Lname	Ssn	...
Research	5	333445555	...	Franklin	T	Wong	333445555	...
Administration	4	987654321	...	Jennifer	S	Wallace	987654321	...
Headquarters	1	888665555	...	James	E	Borg	888665555	...

Figure 6.6
Result of the JOIN operation

Some properties of JOIN

- Consider the following JOIN operation:
  - R(A1, A2, . . ., An) $\bowtie_{R.Ai=S.Bj}$ S(B1, B2, . . ., Bm)
  - Result is a relation Q with degree n + m attributes:
    - Q(A1, A2, . . ., An, B1, B2, . . ., Bm), in that order.
  - The resulting relation state has one tuple for each combination of tuples—r from R and s from S, but *only if they satisfy the join condition* r[Ai]=s[Bj]
  - Hence, if R has n_R tuples, and S has n_S tuples, then the join result will generally have *less than* n_R * n_S tuples.
  - Only related tuples (based on the join condition) will appear in the result

Some properties of JOIN

- The general case of JOIN operation is called a Theta-join: R S
  theta
- The join condition is called *theta*
- *Theta* can be any general boolean expression on the attributes of R and S; for example:
  - R.Ai<S.Bj AND (R.Ak=S.Bl OR R.Ap<S.Bq)
- Most join conditions involve one or more equality conditions “AND”ed together; for example:
  - R.Ai=S.Bj AND R.Ak=S.Bl AND R.Ap=S.Bq

Binary Relational Operations: EQUIJOIN

- EQUIJOIN Operation
- The most common use of join involves join conditions with *equality comparisons* only
- Such a join, where the only comparison operator used is =, is called an EQUIJOIN.
  - In the result of an EQUIJOIN we always have one or more pairs of attributes (whose names need not be identical) that have identical values in every tuple.
  - The JOIN seen in the previous example was an EQUIJOIN.

Binary Relational Operations: NATURAL JOIN Operation

- NATURAL JOIN Operation
  - Another variation of JOIN called NATURAL JOIN — denoted by * — was created to get rid of the second (superfluous) attribute in an EQUIJOIN condition.
    - because one of each pair of attributes with identical values is superfluous
  - The standard definition of natural join requires that the two join attributes, or each pair of corresponding join attributes, *have the same name* in both relations
  - If this is not the case, a renaming operation is applied first.

Binary Relational Operations NATURAL JOIN (contd.)

- Example: To apply a natural join on the DNUMBER attributes of DEPARTMENT and DEPT_LOCATIONS, it is sufficient to write:
  - DEPT_LOCS $\leftarrow$ DEPARTMENT * DEPT_LOCATIONS
- Only attribute with the same name is DNUMBER
- An implicit join condition is created based on this attribute:

  DEPARTMENT.DNUMBER=DEPT_LOCATIONS.DNUMBER

- Another example: Q $\leftarrow$ R(A,B,C,D) * S(C,D,E)
  - The implicit join condition includes *each pair* of attributes with the same name, “AND”ed together:
    - R.C=S.C AND R.D.S.D
  - Result keeps only one attribute of each such pair:
    - Q(A,B,C,D,E)

Example of NATURAL JOIN operation

(a)

PROJ_DEPT

Pname	Pnumber	Plocation	Dnum	Dname	Mgr_ssn	Mgr_start_date
ProductX	1	Bellaire	5	Research	333445555	1988-05-22
ProductY	2	Sugarland	5	Research	333445555	1988-05-22
ProductZ	3	Houston	5	Research	333445555	1988-05-22
Computerization	10	Stafford	4	Administration	987654321	1995-01-01
Reorganization	20	Houston	1	Headquarters	888665555	1981-06-19
Newbenefits	30	Stafford	4	Administration	987654321	1995-01-01

(b)

DEPT_LOCS

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

Figure 6.7
Results of two NATURAL JOIN operations.
(a) PROJ_DEPT ← PROJECT * DEPT.
(b) DEPT_LOCS ← DEPARTMENT * DEPT_LOCATIONS.

Q.7 write about Relational calculas expression for the following questions.

- Relational calculus is considered to be a **nonprocedural** language.
- This differs from relational algebra, where we must write a *sequence of operations* to specify a retrieval request; hence relational algebra can be considered as a **procedural** way of stating a query.

- The tuple relational calculus is based on specifying a number of tuple variables.
- Each tuple variable usually ranges over a particular database relation, meaning that the variable may take as its value any individual tuple from that relation.
- A simple tuple relational calculus query is of the form

  {t | COND(t)}

  - where t is a tuple variable and COND (t) is a conditional expression involving t.
  - The result of such a query is the set of all tuples t that satisfy COND (t).

The Existential and Universal Quantifiers

- Two special symbols called quantifiers can appear in formulas; these are the universal quantifier (∀) and the existential quantifier (∃).
- Informally, a tuple variable t is bound if it is quantified, meaning that it appears in an (∀ t) or (∃ t) clause; otherwise, it is free.
- If F is a formula, then so are (∃ t)(F) and (∀ t)(F), where t is a tuple variable.
  - The formula (∃ t)(F) is true if the formula F evaluates to true for some (at least one) tuple assigned to free occurrences of t in F; otherwise (∃ t)(F) is false.
  - The formula (∀ t)(F) is true if the formula F evaluates to true for every tuple (in the universe) assigned to free occurrences of t in F; otherwise (∀ t)(F) is false.
- ∀ is called the universal or "for all" quantifier because every tuple in "the universe of" tuples must make F true to make the quantified formula true.
- ∃ is called the existential or "there exists" quantifier because any tuple that exists in "the universe of" tuples may make F true to make the quantified formula true.

Languages Based on Tuple Relational Calculus

- The language **SQL** is based on tuple calculus. It uses the basic block structure to express the queries in tuple calculus:
  - SELECT <list of attributes>
  - FROM <list of relations>
  - WHERE <conditions>
- SELECT clause mentions the attributes being projected, the FROM clause mentions the relations needed in the query, and the WHERE clause mentions the selection as well as the join conditions.
  - SQL syntax is expanded further to accommodate other operations. (See Chapter 8).
- Another language which is based on tuple calculus is **QUEL** which actually uses the range variables as in tuple calculus. Its syntax includes:
  - RANGE OF <variable name> IS <relation name>
- Then it uses
  - RETRIEVE <list of attributes from range variables>
  - WHERE <conditions>
- This language was proposed in the relational DBMS INGRES.

- Another variation of relational calculus called the domain relational calculus, or simply, domain calculus is equivalent to tuple calculus and to relational algebra.
- The language called QBE (Query-By-Example) that is related to domain calculus was developed almost concurrently to SQL at IBM Research, Yorktown Heights, New York.
  - Domain calculus was thought of as a way to explain what QBE does.
- Domain calculus differs from tuple calculus in the type of variables used in formulas:
  - Rather than having variables range over tuples, the variables range over single values from domains of attributes.
- To form a relation of degree n for a query result, we must have n of these domain variables— one for each attribute.

- An expression of the domain calculus is of the form

$$\{ x_1, x_2, \ldots, x_n \mid \mathrm{COND}(x_1, x_2, \ldots, x_n, x_{n+1}, x_{n+2}, \ldots, x_{n+m})\}$$

  - where $x_1, x_2, \ldots, x_n, x_{n+1}, x_{n+2}, \ldots, x_{n+m}$ are domain variables that range over domains (of attributes)
  - and COND is a condition or formula of the domain relational calculus.

- Retrieve the birthdate and address of the employee whose name is ‘John B. Smith’.
- Query :

{uv | (∃ q) (∃ r) (∃ s) (∃ t) (∃ w) (∃ x) (∃ y) (∃ z)
(EMPLOYEE(qrstuvwxyz) and q=’John’ and r=’B’ and s=’Smith’)}

- Ten variables for the employee relation are needed, one to range over the domain of each attribute in order.
  - Of the ten variables q, r, s, . . ., z, only u and v are free.
- Specify the *requested attributes*, BDATE and ADDRESS, by the free domain variables u for BDATE and v for ADDRESS.
- Specify the condition for selecting a tuple following the bar (|)—
  - namely, that the sequence of values assigned to the variables qrstuvwxyz be a tuple of the employee relation and that the values for q (FNAME), r (MINIT), and s (LNAME) be ‘John’, ‘B’, and ‘Smith’, respectively.

Q.8 write the relational calculas expression for the following question.

- Retrieve the name and address of all employ as who work for the research department.
- Find the names of employees who work in all the projects controlled by department number 5.
- Retrieve the birthdate and address of the employee whose name is ‘Johb B.smith’.

Example Query Using Existential Quantifier

- Retrieve the name and address of all employees who work for the ‘Research’ department. The query can be expressed as :

{t.FNAME, t.LNAME, t.ADDRESS | EMPLOYEE(t) and (∃ d)
(DEPARTMENT(d) and d.DNAME=‘Research’ and d.DNUMBER=t.DNO) }

- The only *free tuple variables* in a relational calculus expression should be those that appear to the left of the bar (|).
    - In above query, t is the only free variable; it is then *bound successively* to each tuple.
- If a tuple *satisfies the conditions* specified in the query, the attributes FNAME, LNAME, and ADDRESS are retrieved for each such tuple.
    - The conditions EMPLOYEE (t) and DEPARTMENT(d) specify the range relations for t and d.
    - The condition d.DNAME = ‘Research’ is a selection condition and corresponds to a SELECT operation in the relational algebra, whereas the condition d.DNUMBER = t.DNO is a JOIN condition.

Example Query Using Universal Quantifier

- Find the names of employees who work on *all* the projects controlled by department number 5. The query can be:

{e.LNAME, e.FNAME | EMPLOYEE(e) and ((∀ x)(not(PROJECT(x)) or not(x.DNUM=5)

OR ((∃ w)(WORKS_ON(w) and w.ESSN=e.SSN and x.PNUMBER=w.PNO)))))}

- Exclude from the universal quantification all tuples that we are not interested in by making the condition true *for all such tuples*.
  - The first tuples to exclude (by making them evaluate automatically to true) are those that are not in the relation R of interest.
- In query above, using the expression **not(PROJECT(x))** inside the universally quantified formula evaluates to true all tuples x that are not in the PROJECT relation.
  - Then we exclude the tuples we are not interested in from R itself. The expression not(x.DNUM=5) evaluates to true all tuples x that are in the project relation but are not controlled by department 5.
- Finally, we specify a condition that must hold on all the remaining tuples in R.

((∃ w)(WORKS_ON(w) and w.ESSN=e.SSN and x.PNUMBER=w.PNO)

Example Query Using Domain Calculus

- Retrieve the birthdate and address of the employee whose name is ‘John B. Smith’.
- Query :

{uv | (∃ q) (∃ r) (∃ s) (∃ t) (∃ w) (∃ x) (∃ y) (∃ z)

(EMPLOYEE(qrstuvwxyz) and q=’John’ and r=’B’ and s=’Smith’)}

- Ten variables for the employee relation are needed, one to range over the domain of each attribute in order.
  - Of the ten variables q, r, s, . . ., z, only u and v are free.
- Specify the *requested attributes*, BDATE and ADDRESS, by the free domain variables u for BDATE and v for ADDRESS.
- Specify the condition for selecting a tuple following the bar (|)—
  - namely, that the sequence of values assigned to the variables qrstuvwxyz be a tuple of the employee relation and that the values for q (FNAME), r (MINIT), and s (LNAME) be ‘John’, ‘B’, and ‘Smith’, respectively.