19.2.2019

# UNIT-2

## Relational data model and language.

" A relational database consists of a collection of tables, each of which is assigned a unique name. A row in a table represents a relationship among a set of values".

An important feature of relational system that a single database can be spread across several tables.

## Properties of relational table.

1) Values are ~~atomic~~ atomic (sigle value)
2) Column value are of same kind (domain constant)
3) Each column have a unique name.
4) Each row is unique.

## Difference

| DBMS | RDBMS |
|---|---|
| ① In DBMS relationship b/w two tables or files are maintained programically | ① In RDMS relationship b/w two tables or files can be specify at the time of table creation |

| | |
|---|---|
| ① DBMS does not support client server architecture | ② Most of the RDBMS support client server architecture |
| ③ DBMS does not support distributed data bases | ③ Most of the RDBMS support distributed databases. |
| ④ In DBMS there is no high security of data | ④ In RDBMS there are multiple levels of security i.e,- (a) command level (b) object level (c) login in at os level. |
| ⑤ Each table is given as extension in DBMS | ⑤ Many tables are grouped in one data base in RDBMS |

## Merits and Demerits of relational approach

⇒ easy to use.
⇒ Flexibility
⇒ security control an authorization can implement more easily
⇒ Data independence is more easily with organization structure used in relational table/system

⇒ DML → (a) Relational database system is machine performance
(b) if the no: of tables between which relationship is to be established are large and voluminous, the performance of query is definately degraded.

**Integrity constraint** :- An integrity constraint is the condition that can be applied on a data base schema to restrict the data according to the need. If the condition is satisfied, then it can be stored in the database. These integrity constraint can be applied on the database when the DBA of a end user defines the database of the schema.

The DBMS check these constraints when the database application is run

The purpose of these constraints is to

ensure that there should not be any loss in data consistency due to change made to the database by the authorised user. It is of two type

① Entity integrity constraint.
② Referencial integrity constraint

1) Entity integrity constraint

It states that no primary key value can be null. This is because the primary key value is used to identify individual types in a relation, having null values for the primary keys implies that we cannot identify some tuples

For e.g, If two or more tuples had null for their primary keys, we might not be able to distinguish them

2) Referential integrity constraints.

To ensure that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation. This condition called referential integrity

Referential integrity is a property of data which when satisfied, requires every value of one attribute (column) of a relation (table) to exist as a value of another attribute in a different (or the same) relation (table)

| c_id | c_name |
|---|---|
| 1 | |
| 2 | |
| 3 | |

| c_id | c_branch | c_subje |
|---|---|---|
| 2 | | |
| 3 | | |
| 4 | | |

For referential integrity to hold in a relational database, any field in a table is declared a foreign key, can contain only values from a parent tables primary key or a candidate key. For instance, deleting a record that contains values here to buy a foreign key in another table would break

Benefits

→ Improve data quality.
→ Faster development
→ consistency across applications.

3) Domain constraints

Domain constraints specify that within each tuple, the value of each attribute 'A' must be an atomic value from the domain item (A). A domain 'D' is a set of atomic values. A common method of specifying a domain is to specify a data type from which the data values forming the domain are drawn.

Example- set of phone_no, student-roll_no.

The preceding are logical definition of domains. A data type or format is also specified for each domain

Example- The data type of employee - age is an integer no. between 18 to 80. For academic - department - name the data type

all character strings that represent valid department name

4) Key constraints

For the relation schema $R(A_1, A_2 \ldots A_n)$ the key constraint key(K), where 'K' is a subset of 'R' is the restriction that no two tuples of relation 'R' define and the relational schema are have the same values for the attributes in 'K'.

Relational Algebra and calculus.

⇒ The basic set of operations for the relational model is relational algebra.
⇒ A sequence of relational algebra operations forms a relational algebra expression.

Q Why relational algebra is very imp?

① it provides a formal foundation for relational model operations
② It is used as a basis for implementing and optimizing queries in RDBMS.
③ some of its concepts are incorporated into the SQL (structured query lang)

for RDBMS

## Relation Calculus

provides a higher level declarative notation for specifying relational queries

A relation calculus expression creates a new relation which is specified in terms of variable that range over rows of the stored database relation (in tuple calculus) or over column of the stored database (in domain calculus)

### Difference

* In calculus expression, there is no order of operation to specify how to retrieve the query result, it specifies only what information the result should contain.
* $\sigma \rightarrow$ symbol for select
  - the operation applied to each tuple individually, the selection condition does not

than one tuple
- operation is unary i.e. it is applied only over single relation
- select operation is commutative i.e,

$$\sigma_{(cond_1)}(\sigma_{(cond_2)}(R)) = \sigma_{(cond_2)}(\sigma_{(cond_1)}(R))$$

(select) $\sigma \rightarrow$ rows
(project) $\pi \rightarrow$ columns

### Fundamental relational algebra operation

The select operation select tuples that satisfy a given predicate. The predicate appears as the subscript to $\sigma$.

Select operation ($\sigma$) ↓↑ Syntax: $\sigma$ condition/predicate (table name)

loan

| loan_number | branch_no | amount |
|---|---|---|
| L-1 | A | 900 |
| L-2 | B | 1500 |
| L-3 | B | 1500 |
| L-4 | D | 1500 |
| L-5 | E | 1000 |
| L-6 | F | 7000 |

$\sigma \rightarrow$

Select those tuples of the loan relation where the branch is "B"

$\sigma_{branch\_name = 'B'}(Loan)$

| L_no. | Branch_no. | Amount |
|---|---|---|
| | | 1500 |
| L-2 | B | |
| L-3 | B | 1500 |

one, we can find all tuples in which the amount is equals to 1500

$\sigma_{amount=1500}(loan)$

In general we allow comparisons $=, \neq, <, \leq, >, \geq$ in the relation predicate

Further more we can combine several predicate into a largest predicate by using the connectives and $\wedge, \vee$ & $\overline{(NOT)}$

To find those tuples pertaining to loans of more than 1200 made by the 'B' branch.

$\sigma_{Branch\ name='B' \wedge amount>1200}(loan)$

The selection predicate may include comparison between two attributes.

Project operation → Retrieve the data of one or more column
→ if any duplicate data is available it consider it as only one time. (eg → name).

⇒ Suppose we want to list all loan numbers and the amount of the loans, but do not care about the branch name. The project operation allow us to produce this relation

⇒ The project operation is a unary operation that returns an argument relation, with certain attributes left out. Since a relation is a set any, duplicate rows are eliminated.

| loan_no. | Amount |
|---|---|
| $L_1$ | 900 |
| $L_2$ | 1500 |
| $L_3$ | 1500 |
| $L_4$ | 1500 |
| $L_5$ | 1000 |
| $L_6$ | 7000 |

$\pi_{loan\ number,\ amount}(loan)$ ⇒

| loan_no | Amount |
|---|---|
| | |

$\pi_{amount}(loan)$ ⇒

| amount |
|---|
| |

Example :-

$\Pi_{C\_name}$(Depositor) customer_relation

loan_relation

| C_name | Acc_no. |
|---|---|
| Johnson | A-101 |
| Smith. | A-215 |
| Hayes. | A-102 |
| Turner | A-305 |
| Johnson | A-201 |
| Johns. | A-217 |
| Lindsay | A-222 |

| B-name | Loan_no. | Amou |
|---|---|---|
| Down town A | L-17 | 1000 |
| Redwood | L-23 | 2000 |
| Perry-C -ridge | L-15 | 1500 |
| Downtown A | L-14 | 1500 |
| Mianus | L-13 | 500 |
| Round- -Hill E | L-11 | 900 |
| Perry Ridge C | L-16 | 1300 |

Depositor relation

| C_name | loan_no |
|---|---|
| Johns | L-17 |
| Smith | L-23 |
| Hayes | L-15 |
| Jackson | L-14 |
| Curery | L-93 |
| Smith | L-11 |
| William | L-17 |
| Adama | L-16 |

Borrow relation

(*) Union, Set difference and Set intersection can only apply on the set which are compatible to each other.

(Same Column and Same domain)

Union operation. If we queried to find the name of all bank customers who have either an account and loan or both

eg:2 $(\Pi_{customer\_name(borrower)} \cup \Pi_{customer\_name(depositor)})$

commutative

| C_name |
|---|
| Johson |
| Smith |
| Hayes |
| Turner |
| Johns. |
| Lindsay |
| Jackson |
| Curery |
| Williams |
| Adama. |

R ∪ S, where R and S are relation

Rule:

① No. of Columns must be same.

② Domain of every Column must be same. (means numeric or character) → order should be same

eg:1

Student

| Roll no. | Name |
|---|---|
| 1 | A |
| 2 | B |
| 3 | C |

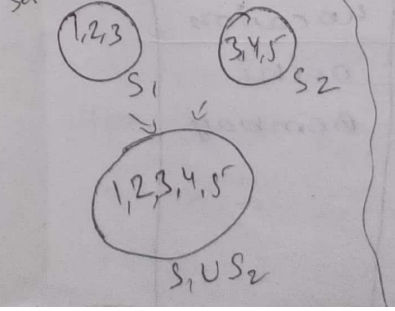
Employee

| Emp. no. | Name |
|---|---|
| 7 | E. |
| 1 | A |

(Student) ∪ (Employee)

| Roll no. | Name |
|---|---|
| 1 | A |
| 2 | B |
| 3 | C |
| 7 | E. |

Set theory




(int, char) can always be union with (int, char) but not (char, int).

25th Feb, 2014

$(A-B) = A \cap B$

not commutative

$A-B \neq B-A$

Set difference operation :- (-).

$S_1$ (123) $S_2$ (3,4)

$S-S_2$ (1,2)

$S_2-S_1$ (4)

fundamental operator

In depositor and borrower

(Relation depositor) − (borrower relation)

= Jonson, Turner, Linda, Vinay.

Q: Name of person who is not a student but not employee → $\pi_{name}$(Student) − $\pi_{name}$(Employee)

Cartesian product operation (×)

Ans → B, C

C+C = Total column

R×R = Total Row

at least one column should be same

Set intersection operation

not a fundamental operator

$A \cap B = A-(A-B)$

Example.

Employee

| E-no. | E-name | designation | salary | department number |
|---|---|---|---|---|
| 1001 | Akshay | clerk | 5000 | 1 |
| 1002 | Ashish | sales person | 5500 | 1 |
| 1003 | Spanish | sales person | 6000 | 2 |
| 1004 | Meena | P.R.O | 6500 | 2 |
| 1006 | Rahul | Manager | 11000 | 2 |

Dept_Relation

| Dept_no. | Dept_name | location |
|---|---|---|
| 1 | account | Delhi |
| 2 | admin | Bombay |

Ques Give all information of employees of department no. 2.

$\sigma_{dept\_no = "2"}$(employee).

Ans

$\sigma_{dept\ no = "2"}$ (Employee)

Q. Give all information for employees who are not in department no. 2.

Ans $\sigma_{dept\ no \neq "2"}$ (Employee)

Q Give the information of all those employees who are in dept no. 1 and who are getting salary more than 5000.

Ans $\sigma_{dept\ no = "1" \wedge salary > 5000}$(Employee).

Join operation

The join operation denoted by '⋈' 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 allow us to process relationships among relations.

The general form of a join operation on two relations

R(A1, A2 --- An)

S(B1, B2 --- Bn) is

R ⋈ <join condition> S

It has four types.

1. Natural join
2. Outer join
3. Equi join
4. Theta join.

1. **Natural join** The natural join is a binary operation that allow us to combine certain selections and one cartesian product into one operation. It is denoted by symbol '⋈'. The natural join operation forms a cartesian product of its two arguments, performs a selection forcing equality on those attributes that appear in both relation schems and finally removes duplicate attributes. It is also denoted by the symbol '*'.

Employee

| E-code | E-name |
|---|---|
| E-001 | Hari |
| E-002 | Om |
| E-003 | Smith |
| E-004 | Jai |

Salary

| E-code | Salary |
|---|---|
| E-001 | 2000 |
| E-002 | 5000 |
| E-003 | 7000 |
| E-004 | 1000 |

emp '⋈' salary.

| E-code | E-name | Salary |
|---|---|---|
| E-001 | Hari | 2000 |
| E-002 | Om | 5000 |
| E-003 | Smith | 7000 |
| E-004 | Jai | 1000 |

(a) Display the name of all employees with salary

Ans. $\pi_{employee\_name,\ salary}$ (emp.'⋈' salary).

26th Feb, 2014.

**Outer join** :- The outer join operation is an extension of the join operation to deal with missing op. information.

Employee

| E-name | street | city |
|---|---|---|
| Ram | Civil line | Mumbai |
| shyam | Park street | Kolkata |
| Rani | M.G street | Delhi |
| Rohit | Nehru Nagar | Hyde-Abad |

Fact_work

| E-name | Branch name | Salary |
|---|---|---|
| Ram | Mesa | 1000 |
| shyam | Wipro | 2000 |
| Kuber | HCL | 3000 |
| Rohit | TCS | 5000 |

employee ⋈ Fact_work.

| E-name | street | city | Branch name | Salary |
|---|---|---|---|---|
| Ram | Civil line | Mumbai | Mesa | 1000 |
| shyam | Park street | Kolkata | Wipro | 2000 |
| Rohit | Nehru Nagar | Hyderabad | TCS | 5000 |

from the above result we observe that we have lost some information to avoid such loss of information we can use the concept of outer join. It is basically of three types.

① Left outer join '⟕'
② Right outer join '⟖'
③ Full outer join '⟗'

① Left outer join :- It takes all the tuple in the left relation that did not match with any tuple in the right relations, pads the tuples with null values for all other attributes from the right relation and adds them to the result of the natural join

employee ⟕ fact_work

| E-name | street | city | Branch name | Salary |
|---|---|---|---|---|
| Ram | Civil line | Mumbai | Mesa | 1000 |
| shyam | Park street | Kolkata | Wipro | 2000 |
| Rohit | Nehru Nagar | Hyderabad | TCS | 5000. |
| Rani | M.G. street | Delhi | Null | Null. |

② Right outer join :-

| E-name | street | city | Branch name | Salary. |
|---|---|---|---|---|
| Ram | Civil line | Mumbai | Mesa | 1000 |
| shyam | Park street | Kolkata | Wipro | 2000 |
| Rohit | Nehru Nagar | Hydera-bad | TCS | 5000 |
| Kuber | NULL | NULL | HCL | 3000 |

employee '⟖' fact work.

## Full outer join

employee ⟗ fact-work.

| E_name | Street | City | Branch name | Salary |
|---|---|---|---|---|
| Ram | Civil line | Mumbai | MESA | 1000 |
| Shyam | Park street | Kolkata | Wipro | 2000 |
| Rohit | Nehru Nagar | Hyderabad | TCS | 5000 |
| Rani | M.G. street | Delhi | NULL | NULL |
| Kuber | NULL | NULL | HCL | 3000 |

③ **Equi join**

The most common use of join involves join condition with equality comparison only so such a join where the only comparision operator (=) is used is called an equi join. In the result of an equi join, we always have one or more pair of attributes that have identical value in every tuple.

④ **Theta join**

A general join condition is of the form

<condition> AND <CONDITION> AND ——— AND <CONDITION>

where each condition is form $A_i \theta B_j$, $A_i$ is an attribute of 'R', $B_j$ is the attribute of 'S', $A_i$ and $B_j$ have same domain and $\theta$ is one of the comparison operations. ($=, \neq, >, \geq, <, \leq$)

A join operation with such a general join condition is called a theta join.

**Query**:- consider the following schema:

SUPPLIER (SUP SUPPLIER_ID, SUPPLIER_NAME, SUPPLIER_ADDERESS).

PARTS (PARTS_ID, PARTS_NAME, PARTS_COLOUR)

CATALOG (SUPPLIER_ID, PART_ID, COST).

write the following queries in relational algebra in SQL.

① Find the name of the supplier whose supply yellow parts.

② Find the name of suppliers who supply both blue and green parts.

③ Find the name of supplier who supply all parts.

SOL:- (1) select supplier_name.

from supplier, parts, catalog.

where supplier.supplier_id = catalog.

where
supplier.supplier_id = catalog.supplier_id
and part.part_id = catalog.part_id
and part_colour = YELLOW.

(2) Select SUPPLIER_NAME
from SUPPLIER, PARTS, CATALOG
where
SUPPLIER.SUPPLIER_ID = CATALOG.SUPPLIER_ID
and PART.PART_ID = CATALOG.PART_ID.
and PART_COLOUR = "BLUE".

Intersect.

SELECT SUPPLIER_NAME.
from SUPPLIER, PARTS, CATALOG.
where.
SUPPLIER.SUPPLIER_ID = CATALOG.SUPPLIER_ID
and PART.PART_ID = CATALOG.PART_ID.
and PART_COLOUR = "GREEN".

(2) select SUPPLIER_NAME
from SUPPLIER, PARTS, CATALOG.
where,
SUPPLIER.SUPPLIER_ID = CATALOG.SUPPLIER_
and PART.PART_ID = CATALOG.PART_ID.
and PART.COLOUR = PART (select all colour
from parts)
or

Divide operation Division operation denoted by '÷' is used to query that include the phrase "for all" as requires the use of one single colourn table and one two columns table.

SSN_PNOS

| ESSN | P_NO. |
|---|---|
| 123456789 | 1 |
| 123456789 | 2 |
| 66688444 | 3 |
| 453453453 | 1 |
| 453453453 | 2 |
| 33344555 | 2 |
| 333 44555 | 3 |

S_PNOS

| PNO |
|---|
| 1 |
| 2 |

SSN_PNOS / S_PNOS =

| SSN |
|---|
| 123456789 |
| 453453453. |

Relation algebra.

(a) $\pi$ SUPPLIER_NAME (PARTS
(supplier ∞ catalog ∞ ($\sigma$ color = "YELLOW"

(2) $\pi$ SUPPLIER_NAME
(supplier ∞ catalog ∞ parts ÷ $\pi$ COLOUR
($\sigma$ colour = 'Blue' and color = 'Green')(PARTS

(3) $\pi$ SUPPLIER_NAME, COLOUR
($\pi$ colour (supplier ∞ catalog ∞ parts) ÷
$\pi$ colours (parts)

Consider the following scheme for

PROJECT DATABASE

PROTECT (P_NO. P_NAME P_MANAGER)

EMPLOYEE

EMPLOYEE (E_no., E_name)

ASSIGNED_TO (Project_no., Employee_no).

Ques) Write SQL - DDL statement for implementation of project database. The SQL statement should clearly indicate the primary key and the foreign key.

Ques) Write the following queries in Relational algebra & SQL

(a) Get the details of employees working on both projects $P_1$ & $P_2$

(b) List the name of employess working on project $P_1$ but not on project $P_2$

(c) Delete the record of employee whose e_no. is $E_1$

(d) List the name of employees who are working on a project for which $E_1$ is the project manager.

Sol:- (1) Create table project (P_NO. NUMBER (3) PRIMARY KEY, P_NAME varchar2(30); P_MANAGER varchar2(30);)

Create table employee (E_NO. NUMER (4) PRIMARY KEY, E_NAME varchar2(30);)

Create table ASSIGNED_TO (PL NO. NUMBER. (4) PRIMARY KEY, E_NO NUMBER (4). FOREIGN KEY).

Ques)(2)(a) *Select E_no, E_name from project, employee, assigned_to where project no is equals to ($P_1$ & $P_2$).

(b) *select E_no, E_name from project, employee, assigned_to where project no is equal to $P_1$