**UNIT – II**

***RELATIONAL DATA MODEL***

**Basic Concepts**

There are three basic conventions

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Item  Customer  Shipment   |  |  | | --- | --- | |  | Entity | |  | Attribute | |  | Relationship | | | | | |
| **S. No** | **Basic Concepts** | |
| 1. | **Entity Set** | * An Entity has set of properties, and values for some set of properties may uniquely identify an entity. * **Ex: *Item & Customer*** * An Entity set is a collection of entities having a same properties |
| 2. | **Attributes** | * The Properties that describe an entity are called attributes. * **Ex :** In the Customer Entity, ***Cust Code, Cust Name, Cust Category, Cust Country*** |
| 3. | **Relationship** | * Relationship is an association among the several Entities. * Relationship Set is a set of relationship of the same type. * **Ex:** Relationship set ***Borrower*** denotes the association between ***Customer*** and ***Loan*** entities set.   Borrower  Customer  Loan |

**Basic Structure of Relation Model**

* The relational model represents the database as a collection of relations.
* A relational database consists of a collection of Tables, each of which is assigned a unique name.

***Relation***

***Attributes***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EMPLOYEE** | **Empno** | **Empname** | **Salary** | **Designation** | **Dno** |
| ***Tuples*** | 1001 | Pramila | 40000 | General Manager | 101 |
| 1002 | Ananthi | 35000 | Project Leader | 102 |
| 1003 | Kumar | 30000 | Team Leader | 103 |
| 1004 | Pradeepa | 25000 | Programmer | 104 |
| 1005 | Praveen | 25000 | Programmer | 104 |

**Terminology**

|  |  |  |
| --- | --- | --- |
| **S. No** | **Informal Terms** | **Formal Terms** |
| 1. | Table | Relation ***(R)*** |
| 2. | Column Header/ Field | Attributes ***(A)*** |
| 3. | All Possible Column Values | Domain ***(D)*** |
| 4. | Rows | Tuples ***(t)*** |
| 5. | Table definition | Schema of Relation |

***Notes:***

***Degree of relation*** : No. of attributes in Relation.

***Cardinality of Relation*** : No of Tuples in Relation.

* A relation schema **R**, denoted by **R(A1,A2,A3,…..An)**is made up of a relation name **R** and the list of attributes **A1,A2,A3,…..An.** Each Attributes **Ai** is the name of a role played by some domain **D** inthe relation schema **R.**
* A relation (or Relation state) **r** of the relation schema **R(A1,A2,A3,…..An),** Also denoted by **r(R),** is a set **n-tuples r={t1,t2,t3,…..,tm}.**
* For all relation **r,** the domain of all attributes of **r** is atomic. A domain is atomic. If elements of the domain are considered to be indivisible units.

**Database Schema**

|  |  |  |
| --- | --- | --- |
| **Database Schema** | **Data instance** | **Example** |
| In which logical design of the database is a Database Schema | In which is a snapshot of the data in the database at a given instance in time. | |  |  |  |  | | --- | --- | --- | --- | | **EMP** | **Empno** | **Empname** | **Dno** | |  | 1001 | Pramila | 101 | | 1002 | Ananthi | 102 | | 1003 | Kumar | 103 | | 1004 | Pradeepa | 104 | | 1005 | Praveen | 104 | |
| **Relation Schema** | **Relation instance** | **Example** |
| The Relation Schema corresponds to the programming language notion of type definition | The Relation instance corresponds to the programming language notion of the variables. The value of a given variable may change with time. Similarly the content of the relation instance may change with time as the relation is updated. | ***Notes:***   1. The convention of using ***lowercase*** names of the relation. 2. Names beginning with an ***uppercase*** letter for the relation schema.   ***Account\_schema = (account\_number, branch\_name, balance)***  We denote the fact the ***account*** is a relation on ***Account\_schema*** by  ***Account (Account\_schema)*** |

**KEYS**

* Keys allow us to identify a set of attributes and thus distinguished entities from each other. Keys also help to uniquely identify relationships, and thus distinguish relationship from each other.

Different types of keys are:

|  |  |  |  |
| --- | --- | --- | --- |
| 1. | **Super Key** | 3. | **Primary Key** |
| 2. | **Candidate Key** | 4. | **Foreign Key** |

1. **Super Key**

* It is a Simple Key.
* It is defined as a set of attributes within a table that can uniquely identify each record with in table.
* Super key is a super set of Candidate Key.

*Eg:*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Student** | **Stdid** | **Name** | **Phone** | **Age** |
|  | 001 | Akhil | 9876543298 | 17 |
|  | 002 | Akhil | 8765439890 | 19 |
|  | 003 | Bhanu | 6398764321 | 18 |
|  | 004 | Priya | 8723456789 | 19 |

1. **Student(stdid)**

* ***Stdid*** is unique in every row of data; hence it can be used to identify each row uniquely.

|  |
| --- |
| **Stdid** |
| 001 |
| 002 |
| 003 |
| 004 |

1. **Student(stdid, Name)**

* ***Name*** of the student to be same, but their ***stdid*** cannot be same. Hence this is combination is also be a super key.

|  |  |
| --- | --- |
| **Stdid** | **Name** |
| 001 | Akhil |
| 002 | Akhil |
| 003 | Bhanu |
| 004 | Priya |

1. **Student(Phone)**

* ***phone*** number also be unique. Hence it can also be a Super Key.

|  |
| --- |
| **Phone** |
| 9876543298 |
| 8765439890 |
| 6398764321 |
| 8723456789 |

1. **Candidate Key**

* A super key without redundancy.
* It is not divided further
* Minimum set of attributes used to uniquely differentiate record of the table. A candidate key can never be NULL or empty.
* Its value should be also unique.
* There can be more than one candidate key for a table.
* A Candidate key can be combination of more than one attributes.

*Eg:*

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Student 1** | **Stdid** | **Name** | **Mark** |  | **Student 2** | **Stdid** | **Name** | **Mark** |
|  | S1 | Akhil | 40 |  |  | S1 | Akhil | 40 |
|  | S2 | Akhil | 40 |  |  | S2 | Bhanu | 50 |
|  | S3 | Bhanu | 50 |  |  | S3 | Akhil | 90 |
|  | S2 | Bhanu | 50 |  |  | S4 | Priya | 50 |

1. **Student1(Stdid,Name)**

* Combination of minimum attribute on that must have unique values.

|  |  |
| --- | --- |
| **Stdid** | **Name** |
| S1 | Akhil |
| S2 | Akhil |
| S3 | Bhanu |
| S2 | Bhanu |

1. **Student(Stdid)**

* On table ***Student2*** contain ***stdid*** only unique values so ***Stdid*** only is a Candidate key

|  |
| --- |
| **Stdid** |
| S1 |
| S2 |
| S3 |
| S4 |

1. **Primary Key**

* It is a Candidate key that is most appropriate to become the main key for any table.
* It is a key that can uniquely identify each record in a table.
* Uniquely values must never be NULL.
* Uniquely identify each record in table.
* Primary Key is a mandatory for every table. Each record must have a value for its primary key.
* When choosing a primary key fro, the pool of candidate key always choose a single simple key over a composite key

Eg:

1. **Student2(Stdid, Name, Mark)**

* On table ***Student2*** in this ***Stdid*** is a Primary Key

1. **Foreign Key**

* Foreign key is an attribute which establish relationship between two tables.
* Foreign key is a field (or Collection of attribute) in one table that refers to the primary key in another table.
* Records cannot be inserted into a detailed table (Child Table) if corresponding record in the Master table (Parent Table) does not exist.
* Record of a Master Table (Parent table) cannot be deleted if corresponding record in the detailed table (Child Table) actually exist.
* Parent must be unique or Primary Key
* Child may have duplicated as well as a NULL unless it is specified.
* Constraints specified on Child not in Parent.
* Parent record can be deleting only if no child record exist.
* Parent cannot modify if child record exists.

Eg:

***Parent or Master or Reference Table*** ***Child Table or Detailed Table***

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Person** | **Pid** | **Name** | **Phone** |  | **Order** | **Oid** | **Ono** | **Pid** |
|  | P1 | Akhil | 9876543298 |  |  | O1 | 1234 | P3 |
|  | P2 | Bhanu | 8765439890 |  |  | O2 | 5678 | P3 |
|  | P3 | Akhil | 6398764321 |  |  | O3 | 8908 | P1 |
|  | P4 | Priya | 8723456789 |  |  | O4 | 3457 | P2 |

* Pid in Order table is a Foreign Key

**QUERY LANGUAGES**

* Query language is a language in which user requests information from the database.
* ***Categories:***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| i. | **Procedural** | | Eg | **Relational Algebra** |
| ii. | **Non - Procedural** | | Eg | **Relational Calculus** |
| **Procedural** | | * The user instructs the system to perform a sequence of operations on the database to compute the desire result   Eg : **Relational Algebra** | | | | **Relational Algebra**   1. Fundamental Operation 2. Additional Operation 3. Extended Operation |
| **Non - Procedural** | | * The user describes the desired information without giving a specific procedure for obtaining that information.   Eg: **Relational Calculus** | | | | **Relational Calculus**   1. The Tuples Relational Calculus 2. The Domain Relational Calculus |

**RELATIONAL ALGEBRA**

1. **FUNDAMENTAL RELATIONAL – ALGEBRA OPERATIONS**

|  |  |  |  |
| --- | --- | --- | --- |
| **Unary Operations** | | **Binary Operations** | |
| 1. | **SELECT** | 4. | **UNION** |
| 2. | **PROJECT** | 5. | **SET DIFFERENCE** |
| 3. | **RENAME** | 6. | **CARTESTIAN PRODUCT** |

**DATA SCHEMA**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ACCOUNT** |  |  | | |
|  |  |  | | |
| **account-no** | branch-name | balance | | |
| **BRANCH** |  |  | | |
|  |  |  | | |
| **branch-name** | branch-city | Assets | | |
| **CUSTOMER** |  | |  | | | |
|  |  | |  | | | |
| **customer-name** | customer-street | | customer-city | | | |
| **LOAN** |  | | |  | |
|  |  | | |  | |
| **loan-number** | branch-name | | | Amount | |

|  |  |  |  |
| --- | --- | --- | --- |
| **ACCOUNT** | **account-no** | **branch-name** | **balance** |
|  | A-101 | Downtown | 500 |
|  | A-102 | Perryridge | 400 |
|  | A-201 | Brighton | 900 |
|  | A-215 | Mianus | 700 |
|  | A-217 | Bighton | 750 |
|  | A-222 | Redwood | 700 |
|  | A-305 | Round Hill | 350 |

|  |  |  |  |
| --- | --- | --- | --- |
| **BRANCH** | **branch-name** | **branch-city** | **assets** |
|  | Brighton | Brooklyn | 7100000 |
|  | Downtown | Brooklyn | 9000000 |
|  | Mianus | Horseneck | 400000 |
|  | North Town | Rye | 3700000 |
|  | Perryridge | Horseneck | 1700000 |
|  | Pownal | Bennington | 300000 |
|  | Redwood | Palo Alto | 2100000 |
|  | Round Hill | Horseneck | 8000000 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CUSTOMER** | **customer-name** | | **customer-street** | | **customer-city** | |
|  | Adams | | Spring | | Pittsfield | |
|  | Brooks | | Senator | | Brooklyn | |
|  | Curry | | North | | Rye | |
|  | Glenn | | Sand Hill | | Woodside | |
|  | Green | | Walnut | | Stamford | |
|  | Hayes | | Main | | Harrison | |
|  | Johnson | | Alma | | Palo alto | |
|  | Jones | | Main | | Harrison | |
|  | Lindsay | | Park | | Pittsfield | |
|  | Smith | | North | | Rye | |
|  | Turner | | Putnam | | Stamford | |
|  | Williams | | Nassau | | Princeton | |
| **DEPOSITOR** | | **customer-name** | | **account-no** | |
|  | | Hayes | | A-102 | |
|  | | Johnson | | A-101 | |
|  | | Johnson | | A-201 | |
|  | | Jones | | A-217 | |
|  | | Lindsay | | A-222 | |
|  | | Smith | | A-215 | |
|  | | Turner | | A-305 | |

|  |  |  |  |
| --- | --- | --- | --- |
| **LOAN** | **loan-number** | **branch-name** | **Amount** |
|  | L-11 | Round Hill | 900 |
|  | L-14 | Downtown | 1500 |
|  | L-15 | Perryridge | 1500 |
|  | L-16 | Perryridge | 1300 |
|  | L-17 | Downtown | 1000 |
|  | L-23 | Redwood | 2000 |
|  | L-93 | Mianus | 500 |

|  |  |  |
| --- | --- | --- |
| **BORROWER** | **customer-name** | **loan-number** |
|  | Adams | L-16 |
|  | Curry | L-93 |
|  | Hayes | L-15 |
|  | Jackson | L-14 |
|  | Jones | L-17 |
|  | Smith | L-11 |
|  | Smith | L-23 |
|  | Williams | L-17 |

1. The **SELECT** Operation **(σ)**

* The **SELECT** operation selects tuples that satisfy a given predicate.
* The lower case Greek letter sigma **(σ)** to denote selection.
* The predicate appears as a subscript to **σ.**
* The argument relation is in parentheses after the **σ.**
* ***Syntax***

***σ <****Selection Condition****> (R)***

**σ – SELECT** Operation

**R –** Relation Name (Table Name)

Eg:

* Select those tuples of the loan relation where the branch is ***“Perriridge”.***

**σ** branch-name**=**”Perriridge”**(LOAN)**

**Result**

|  |  |  |  |
| --- | --- | --- | --- |
| **LOAN** | **loan-number** | **branch-name** | **amount** |
|  | L-15 | Perryridge | 1500 |
|  | L-16 | Perryridge | 1300 |

1. The **Project** Operation **(Π)**

* The **Project** operation allows us to list out the certain condition attributes with domain.
* The **Project** operation is a unary operation that returns its argument relation, with certain attributes left out.
* A relation is a set, any duplicate rows are eliminated.
* Projection is denoted by Upper case Greek Letter **pi Π**
* We list those attributes that we wish to appear in the result as a subscript to **Π.**
* The argument relation follows in parentheses.
* ***Syntax***

**Π** *<Attribute List>* **= (R)**

**Π – Project** operation

R  **–** Relation Name (Table Name)

Attribute List – Column Name.

Eg:

* List the loan-number, amount from the loan relation.

**Π** loan-number, amount **= (LOAN)**

Result

|  |  |  |
| --- | --- | --- |
| **LOAN** | **loan-number** | **amount** |
|  | L-11 | 900 |
|  | L-14 | 1500 |
|  | L-15 | 1500 |
|  | L-16 | 1300 |
|  | L-17 | 1000 |
|  | L-23 | 2000 |
|  | L-93 | 500 |

**Composition of relational operation**

* It is the combination of ***select*** and ***project*** operation.
* **Relational Algebra** operations can be composed together into a **Relational Algebra expression.**
* Relational Algebra expression is just like composing arithmetic operations such as into arithmetic expression.

Eg:

* Find those customers who live in ***“Rye”.***

**Π** customer-name (**σ** customer-city = “Rye” **(CUSTOMER))**

Result

|  |  |
| --- | --- |
| **CUSTOMER** | **customer-name** |
|  | Curry |
|  | Smith |

1. The **UNION** operation **()**

* The result of this operation, denoted **R 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.
* A union operation RS to be valid, we require that two conditions hold:
  1. The relation **R** and **S** must be of the same arity. That is, they must have the same number of attributes.
  2. The domains of the *i* th attribute of **R** and the *i* th attribute of **S** must be same for all *i.*
* ***Syntax***

**R S**

Eg:

* Find the names of customers who have either an account or a loan or both.

1. Names of all customer with a loan is **Π** customer-name (BORROWER)
2. Names of all customer with account is **Π** customer-name (DEPOSITOR)

**(Π** customer-name (BORROWER))  **(Π** customer-name (DEPOSITOR))

Result

|  |
| --- |
| **customer-name** |
| Adams |
| Curry |
| Hayes |
| Johnson |
| Jones |
| Lindsay |
| Smith |
| Turner |
| Williams |

1. The **SET DIFFERENCE** operation **()**

* It is denoted by **().**
* It allows us to find tuples that are in one relation but are not in another.
* ***Syntax***

**RS**

* A relation containing those tuples in **R** but not in **S.**

Eg:

* **(Π** customer-name (DEPOSITOR))  **(Π** customer-name (BORROWER))

Result

|  |
| --- |
| **customer-name** |
| Johnson |
| Lindsay |
| Turner |

1. The  **CARTESIAN – PRODUCT** operation **(**×**)**

* It allows us to combine information from any two relations.
* Defined as

**RS = {tq| t r** and **q s}**

* Assume that attributes of **r(R)** and **s(S)** are disjoint.

Eg:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Relation - R** | |  | **Relation – S** | | |  | **R × S** | | | | |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **A** | **B** |  | **C** | **D** | **E** |  | **A** | **B** | **C** | **D** | **E** |
| α | 1 |  | α | 10 | A |  | α | 1 | α | 10 | a |
| β | 2 |  | β | 10 | A |  | α | 1 | β | 10 | a |
|  |  |  | β | 20 | B |  | α | 1 | β | 20 | b |
|  |  |  | γ | 10 | B |  | α | 1 | γ | 10 | b |
|  |  |  |  |  |  |  | β | 2 | α | 10 | a |
|  |  |  |  |  |  |  | β | 2 | β | 10 | a |
|  |  |  |  |  |  |  | β | 2 | β | 20 | b |
|  |  |  |  |  |  |  | β | 2 | γ | 10 | b |

1. The **RENAME** operation

* It allows us to name the relational algebra expressions.
* It allows us to refer to a relation by more than one name.
* The ***RENAME*** operator denoted by the lowercase Greek letter rho **(.**
* ***Expression***

1. **x(E)** returns the expression **E** under the name **X.**
2. **x (A1,A2,A3….An) (E)** returns the result of expression **E** under the name **X,** and with the attributes renamed to A1,A2,A3,….An.

Eg:

* Consider the ***BOOK*** relation with attributes ***title, author, year and price***. The ***Rename*** operator is used on ***BOOK*** relation as follows.

Temp (Bname, Aname, Pyear, Bprice)(BOOK)

Here both the relation name and attribute names is renamed.

Temp (BOOK) – Here only the relation name is renamed.

**Formal definition of the Relational Algebra:**

* A basic expression in the relational algebra consists of either one of the following:
  + A relation in the Database.
  + A constant relation.
* A general expression in the relational algebra is constructed out of smaller sub expression.
* Let **E1** and **E2** be relational-algebra expression.

|  |  |  |  |
| --- | --- | --- | --- |
| **E1** |  | | **E2** |
| **E1** |  | | **E2** |
| **E1** |  | | **E2** |
| **σp (E1)** | | Where **P** is a predicated on attributes in **E1** | | |
| **Π** s**(E1)** | | Where **S** is a list consisting of some of the attributes in **E1** | | |
| **x(E1)** | | Where **X** is the new name for the result of **E1.** | | |

1. **ADDITIONAL RELATIONAL - ALGEBRA OPERATIONS**

* Additional operation that can be expressed in terms of basic operations

|  |  |  |  |
| --- | --- | --- | --- |
| a. | **SET INTERSECTION** | c. | **DIVISION** |
| b. | **NATURAL JOIN** | d. | **ASSIGNMENT** |

* 1. The **SET – INTESECTION** operation
* Notation used is
* Defined as

**RS = {t / t r** and **r s}**

* Assume **R, S** have the same attributes.
* The attributes of **r** and **s** are compatible.

**RS = R – (R – S)**

Eg:

* To find all customer who have both Loan and an account. Using set intersection

**Π** customer-name (BORROWER))  **(Π** customer-name (DEPOSITOR))

Result

|  |
| --- |
| **customer-name** |
| Hayes |
| Jones |
| Smith |

* 1. The **NATURAL JOIN** operation **(|**×**|)**
* The ***Natural Join*** is a binary operation that allows us to combine certain Selections and a Cartesian product into one operation.
* It is denoted by the Join 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 schemas; and finally removes duplicate attributes.

Eg:

* Find the names of all customers who have a loan at the bank, and find the amount of the loan.

**Π** customer-name, loan-number, amount **(BORROWER |×| LOAN)**

Result

|  |  |  |
| --- | --- | --- |
| **customer-name** | **loan-number** | **Amount** |
| Adams | L-16 | 1300 |
| Curry | L-93 | 500 |
| Hayes | L-15 | 1500 |
| Jackson | L-14 | 1500 |
| Jones | L-17 | 1000 |
| Smith | L-11 | 2000 |
| Smith | L-23 | 900 |
| Williams | L-17 | 1000 |

* The Theta Join operation is an extension to the natural-join operation that allows us to combine a selection and a Cartesian product into a single operation.
* The Theta Join **r |×|θ s** isdefined as follows

**r |**×**|θ s = σθ (r** × **s)**

* 1. The **DIVISION** Operation **( ÷ )**
* The Division Operation denoted by , is suited to queries that includes the phrase ***“FOR ALL” & “AT ALL”.***
* Formally let r® and s(S) be relations, and let . That is, every attribute of schema S is also in schema R.
* The relation **r ÷ s** is a relation on schema **R ÷ S.**
* That is, on the schema containing all attributes of schema R that are not in schema S. A tuple t is in **r ÷ s** if and only if both of two condition hold.
* For Example
* Suppose that we wish to find all customers who have an account at ***all*** the branches located in “Brooklyn”.

r1 **= Π branch\_name** **(σbranch\_city = “Brooklyn”(branch))**

Result

|  |
| --- |
| **branch-name** |
| Brighton |
| Downtown |

r2 **= Π customer\_name, branch\_name** **(depositor |×| account)**

Result

|  |  |
| --- | --- |
| **customer\_name** | **branch-name** |
| Hayes | Perryidge |
| Johnson | Downtown |
| Johnson | Brighton |
| Jones | Brighton |
| Lindsay | Redwood |
| Smith | Mianus |
| Turner | Round Hill |

r2 **= (Π customer\_name, branch\_name** **(depositor |×| account))**

**÷ (Π branch\_name** **(σbranch\_city = “Brooklyn”(branch)))**

Result

|  |
| --- |
| **customer-name** |
| Johnson |

* 1. The **ASSIGNMENT** Operation
* A relational algebra expression by assigning parts of it to temporary relation variables.
* The assignment operation, denoted by , works like assignment in a programming language.

Temp1 ΠR-S (r)

Temp2 ΠR-S ((Temp1× s) - ΠR-S,S (r))

Result = Temp1 – Temp2

* The result of the expression to the right of the is assigned to the relation variable on the left of the .

1. **EXTENDED RELATIONAL – ALGEBRA OPERATIONS**

|  |  |  |
| --- | --- | --- |
| **S. No** | **Extended Operations** | **Description** |
| 1 | Generalized Projection | A simple extension is to allow arithmetic operations as part of projection. |
| 2 | Aggregate | As computing the sum of the elements of a set, or their average. |
| 3 | Outer – Join | Which allows relational – algebra expression to deal with Null values, which model missing information. |

1. **Generalized Projection**

* The Generalized projection operation extends the projection operation by allowing arthimatic function to be used in the projection list.
* The generalized projection operation has the form

**Π F1,F2,….Fn(E)**

* Where **E** is any relational-algebra expression, and each of F1, F2 ……. Fn is an arithmetic expression involving constants and attributes in the schema of **E**.
* We can apply the Rename operation to the result of Generalized projection in order to give it a name.
* For Example

The *Credit\_info* Relation as below

|  |  |  |
| --- | --- | --- |
| **customer\_name** | **limit** | **credit\_balance** |
| Curry | 2000 | 1750 |
| Hayes | 1500 | 1500 |
| Jones | 6000 | 700 |
| Smith | 2000 | 400 |

* Find how much more each person can spend.

**Π customer\_name, limit - credit\_balance (credit\_info)**

The attribute resulting from the expression limit- credit\_balance does not have a name.

**Π customer\_name, limit - credit\_balance as credit\_avaliable(credit\_info)**

Result

|  |  |
| --- | --- |
| **customer\_name** | **credit\_avaliable** |
| Curry | 250 |
| Hayes | 0 |
| Jones | 5300 |
| Smith | 1600 |

The second attribute of this generalized projection has been given the name *credit\_avaliable.*

1. **Aggregate Function**

* Aggregate function takes a collection of values and returns a single value as a result.
* The collections on which aggregate functions operate can have multiple occurrence of a value; the order in which the value appear is not relevant. Such collections are called ***Multi set.*** Sets are a special case of multi sets where there is only one copy of each element.
* Avg, min., max., sum, count are few aggregate functions.
* The symbol is the letter in “Calligraphic G”.
* Aggregate operation in relation algebra is defined as

**G1, G2 ……. Gn g F1(A1), F2(A2) ……. Fm(Am) (E)**

* Here E is any relational algebra expression.
* **G1, G2 ……. Gn** is alist of attributes on which to group**.**
* Each **Fi** is an aggregate function.
* Each **Ai** is an attribute name.
* For Example.

The pt\_works relation

|  |  |  |
| --- | --- | --- |
| **customer\_name** | **branch\_name** | **Salary** |
| Adams | Perryidge | 1500 |
| Brown | Perryidge | 1300 |
| Gopal | Perryidge | 5300 |
| Johnson | Downtown | 1500 |
| Loreena | Downtown | 1300 |
| Peterson | Downtown | 2500 |
| Rao | Austin | 1500 |
| Sato | Austin | 1600 |

* To find the total salary sum of the all part-time employees at each branch of the bank separately, rather than the sum for the entire bank.

branch\_name g sum(salary)(pt\_works)

In the expression, the attribute *branch\_name* in the left-hand subscript of **g** indicates that input relation *pt\_works* must be divided into groups based on the values of *branch\_name*.

Result

|  |  |  |
| --- | --- | --- |
| **customer\_name** | **branch\_name** | **Salary** |
| Rao | Austin | 1500 |
| Sato | Austin | 1600 |
| Johnson | Downtown | 1500 |
| Loreena | Downtown | 1300 |
| Peterson | Downtown | 2500 |
| Adams | Perryidge | 1500 |
| Brown | Perryidge | 1300 |
| Gopal | Perryidge | 5300 |

The expression sum(salary) in the right-hand subscript of **g** indicates that for each group of tuples, the aggregation function **sum** must be applied on the collection of values of the salary attribute.

Result

|  |  |
| --- | --- |
| **branch\_name** | **salary** |
| Austin | 3100 |
| Downtown | 5300 |
| Perryidge | 8100 |

1. **Outer Join**

* The Outer Join operation is an extension of the join operation to deal with missing information.
* Computes the join and then adds tuples from one relation that do not match tuples in the other relation to the result of the join.
* Use Null values
  1. Null signifies that the value is unknown or does not exist.
  2. All comparisons involving null are false by definition.
* There are three forms of the operation
  1. Left Outer Join
  2. Right Outer Join
  3. Full Outer Join
* For Example

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Relation of ***loan*** | | |  | Relation of ***borrower*** | |
| **loan\_no** | **branch\_name** | **amount** |  | **customer\_name** | **loan\_no** |
| L-170 | Downtown | 3000 |  | Jones | L-170 |
| L-230 | Redwood | 4000 |  | Smith | L-230 |
| L-260 | Perriridge | 1700 |  | Hayes | L-155 |

1. **Left Outer Join ( )**

* It takes all tuples in the left relation that did not match with any tuple in the right.
* Pads the tuples with null values for all other attributes from the right relation.
* Adds them to result of the natural join.

Result

|  |  |  |  |
| --- | --- | --- | --- |
| ***loan borrower*** | | | |
| **loan\_no** | **branch\_name** | **amount** | **customer\_name** |
| L-170 | Downtown | 3000 | Jones |
| L-230 | Redwood | 4000 | Smith |
| **L-260** | **Perriridge** | **1700** | **Null** |

1. **Right Outer Join ( )**

* It is symmetric with the left outer join.
* It pads tuples from the right relation that did not match any from the left relation with the Null.
* And adds them to the result of the natural join.

Result

|  |  |  |  |
| --- | --- | --- | --- |
| ***loan borrower*** | | | |
| **loan\_no** | **branch\_name** | **amount** | **customer\_name** |
| L-170 | Downtown | 3000 | Jones |
| L-230 | Redwood | 4000 | Smith |
| **L-260** | **Null** | **Null** | **Hayes** |

1. **Full Outer Join ( )**

* It does both of those operations, padding tuples from the left relation that did not match any from the right relation.
* As well as tuples from the right relation that did not match any from the left relation,
* And adds them to the result of the natural join.

Result

|  |  |  |  |
| --- | --- | --- | --- |
| ***loan borrower*** | | | |
| **loan\_no** | **branch\_name** | **amount** | **customer\_name** |
| L-170 | Downtown | 3000 | Jones |
| L-230 | Redwood | 4000 | Smith |
| **L-260** | **Perriridge** | **1700** | **Null** |
| **L-155** | **Null** | **Null** | **Hayes** |

**NULL VALUES**

* The special values, Null indicates ***“Values unknown or Nonexistent”*** any arithmetic operation such as (**+, - , \* , /**) involving null values must return a null result.
* Similar, any comparison ( <, >, ≤ , ≥ , =, ≠ ) involving null value evaluate to special value ***Unknown.***
* We cannot say for sure whether the result of the comparison is true or false, so we say that the result is the new truth values ***unknown***.
* Comparison involving nulls may occur inside Boolean expression involving the ***and, or & not*** operations.
  1. **and :**

|  |  |  |
| --- | --- | --- |
| **True and Unknown** | **=** | Unknown |
| **False and Unknown** | **=** | False |
| **Unknown and Unknown** | ***=*** | Unknown |

* 1. **or :**

|  |  |  |
| --- | --- | --- |
| **True and Unknown** | **=** | True |
| **False and Unknown** | **=** | Unknown |
| **Unknown and Unknown** | ***=*** | Unknown |

* 1. **not :**

|  |  |  |
| --- | --- | --- |
| **Not Unknown** | ***=*** | Unknown |

* Relational operations deal with null vales

|  |  |
| --- | --- |
| **SELECT** | * The selection operation evaluates predicate P in σ p(E) on each tuple t in E. * If the predicate returns the value true, t is added to the result. * Otherwise if the predicate return unknown or false, t is not added to the result. |
| **JOIN** | * Joins can be expressed as a Cartesian product followed by a selection. * Thus, the definition of how selection handles nulls also define how joins operation handle null. |
| **PROJECTION** | * The projection operation treats nulls just like any other values when eliminating duplicates. |
| **UNION, INSERTION, DIFFERENCE** | * These operation treats nulls just as the projection operation does they treat tuples that have the same values on all fields as duplicates even if some of the fields have null values in both tuples. * The behavior is rather arbitrary, especially in the case of intersection and difference, since we do not know if the actual values (if any) represented by the nulls are the same. |
| **GENERALIZED PROJECTION** | * Duplicate tuples containing null values are handles as in the projection. |
| **AGGREGATE** | * When nulls occur in grouping attributes, the aggregate operation treats them just as in projection. * If two tuples are the same on all grouping attributes, the operation places them in the same group, even if some of their attribute values are null |
| **OUTER JOIN** | * It behaves just like join express, except on tuples that do not occur in the join result. * Such tuples may be added to the result, padding with nulls. |

**MODIFICATION OF THE DATABASE**

* The extraction of information from the database. We address how to add, remove or change information the database.
* We express database modifications by the assignment operation.

1. Deletion
2. Insertion
3. Updating

|  |  |  |
| --- | --- | --- |
| **Deletion** | We remove the selected tuples from the database.  **r r - E**  Where  The r is the relation.  E is the Relational algebra Query | **For Example:**  Delete all of Simth’s account records.  **depositor depositor σcustomer-name = (depositor)** |
| **Insertion** | To insert data into a relation, we either a tuple to be inserted or write a query whose result is a set of tuples to be inserted.  **R r E** Where  The R is the relation.  E is the Relational algebra expression. | **For Example:**  To insert the fact that Smith has 1200 in account A – 973 at Perryridge branch.  **account account {(A-973,”Perryridge”,1200)}**  **depositor depositor {(“Simth”,A-973)}** |
| **Updating** | To change a value in a tuple without changing all values in the tuples, the generalized projection operator is used.  r Π F1,F2,… Fn (E) | **For Example :**  To illustrate the use of the update operation, suppose that interest payments are being made, and that all balance is to be increased by 5 percentages.  **account Π account\_number, branch\_name, balance \* 1.05 (account).** |