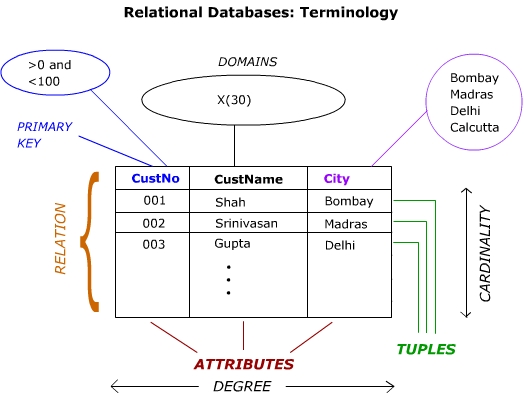
**3.1 The Relational Model**

Relational Databases: Terminology



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Databases: Case Example   Ord\_Aug**   |  |  |  | | --- | --- | --- | | Ord # | OrdDate | Cust# | | 101 | 02-08-94 | 002 | | 102 | 11-08-94 | 003 | | 103 | 21-08-94 | 003 | | 104 | 28-08-94 | 002 | | 105 | 30-08-94 | 005 | | **Ord\_Items**   |  |  |  | | --- | --- | --- | | Ord # | Item # | Qty | | 101 | HW1 | 100 | | 101 | HW3 | 50 | | 101 | SW1 | 150 | | 102 | HW2 | 10 | | 103 | HW3 | 50 | | 104 | HW2 | 25 | | 104 | HW3 | 100 | | 105 | SW1 | 100 | |
| **Items**   |  |  |  | | --- | --- | --- | | **Item #** | **Descr** | **Price** | | HW1 | Power Supply | 4000 | | HW2 | 101-Keyboard | 2000 | | HW3 | Mouse | 800 | | SW1 | MS-DOS 6.0 | 5000 | | SW2 | MS-Word 6.0 | 8000 | | **Customers**   |  |  |  | | --- | --- | --- | | **Ord #** | **OrdDate** | **Cust#** | | 101 | 02-08-94 | 002 | | 102 | 11-08-94 | 003 | | 103 | 21-08-94 | 003 | | 104 | 28-08-94 | 002 | | 105 | 30-08-94 | 005 | |

|  |  |  |
| --- | --- | --- |
| **Term** | **Meaning** | **Eg. from the given Case Example** |
| **Relation** | A table | Ord\_Aug, Customers, Items etc. |
| **Tuple** | A row or a record in a relation. | A row from Customers relation is a Customer tuple. |
| **Attribute** | A field or a column in a relation. | Ord\_Date, Item#, CustName etc. |
| **Cardinality of a relation** | The number of tuples in a relation. | Cardinality of Ord\_Items relation is 8 |
| **Degree of a relation** | The number of attributes in a relation. | Degree of Customers relation is 3. |
| **Domain of an attribute** | The set of all values that can be taken by the attribute. | Domain of Qty in Ord\_Items is the set of all values which can represent quantity of an ordered item. |
| **Primary Key of a relation** | An attribute or a combination of attributes that uniquely defines each tuple in a relation. | Primary Key of Customers relation is Cust#.   Ord# and Item# combination forms the primary Key of Ord\_Items |
| **Foreign Key** | An attribute or a combination of attributes in one relation R1 which indicates the relationship of R1 with another relation R2.   The foreign key attributes in R1 must contain values matching with those of the values in R2 | Cust# in Ord\_Aug relation is a foreign key creating reference from Ord\_Aug to Customers. This is required to indicate the relationship between Orders in Ord\_Aug and Customers.   Ord# and Item# in Ord\_Items are foreign keys creating references from Ord\_Items to Ord\_Aug and Items respectively. |

**3.2 Properties of Relations**

• No Duplicate Tuples – A relation cannot contain two or more tuples which have the same values for all the attributes. i.e., In any relation, every row is unique.  
• Tuples are unordered – The order of rows in a relation is immaterial.  
• Attributes are unordered – The order of columns in a relation is immaterial.  
• Attribute Values are Atomic – Each tuple contains exactly one value for each attribute.

It may be noted that many of the properties of relations follow the fact that the body of a

relation is a mathematical set.

**3.3 Integrity Rules**

**Entity Integrity**

Entity Integrity ensures that each row can be uniquely identified by an attribute called the primary key. The primary key cannot be NULL.

**Domain Integrity**

Domain integrity ensures that only a valid range values is allowed to be stored in a common. It can be enforced by restricting the type of data, the range of values and the format of the data.

**Referential Integrity**

Ensuring that all the values in the foreign key match with the value of the corresponding primary key is called Referential Integrity.

The following are the integrity rules to be satisfied by any relation.

• No Component of the Primary Key can be null.

• The Database must not contain any unmatched Foreign Key values. This is called the referential integrity rule.

Q: Can the Foreign Key accept nulls?  
A: Yes, if the application business rule allows this.

How do we explain this ?

Unlike the case of Primary Keys, there is no integrity rule saying that no component of the foreign key can be null. This can be logically explained with the help of the following example:

**Employee**

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp#** | **EmpName** | **EmpCity** | **EmpAcc#** |
| X101 | Shekhar | Bombay | 120001 |
| X102 | Raj | Pune | 120002 |
| X103 | Sharma | Nagpur | Null |
| X104 | Vani | Bhopal | 120003 |

**Account**

|  |  |  |
| --- | --- | --- |
| **ACC#** | **OpenDate** | **BalAmt** |
| 120001 | 30-Aug-1998 | 5000 |
| 120002 | 29-Oct-1998 | 1200 |
| 120003 | 01-Jan-1999 | 3000 |
| 120004 | 04-Mar-1999 | 500 |

EmpAcc# in Employee relation is a foreign key creating reference from Employee to Account. Here, a Null value in EmpAcc# attribute is logically possible if an Employee does not have a bank account. If the business rules allow an employee to exist in the system without opening an account, a Null value can be allowed for EmpAcc# in Employee relation.

In the case example given, Cust# in Ord\_Aug cannot accept Null if the business rule insists that the Customer No. needs to be stored for every order placed.

The next issue related to foreign key reference is handling deletes / updates of parent?

In the case example, can we delete the record with Cust# value 002, 003 or 005 ?

The default answer is NO, as long as there is a foreign key reference to these records from some other table. Here, the records are referenced from the order records in Ord\_Aug relation. Hence Restrict the deletion of the parent record.

Deletion can still be carried if we use the Cascade or Nullify strategies.

**Cascade:** Delete/Update all the references successively or in a cascaded fashion and finally delete/update the parent record. In the case example, Customer record with Cust#002 can be deleted after deleting order records with Ord# 101 and 104. But these order records, in turn, can be deleted only after deleting those records with Ord# 101 and 104 from Ord\_Items relation.

**Nullify:** Update the referencing to Null and then delete/update the parent record. In the above example of Employee and Account relations, an account record may have to be deleted if the account is to be closed. For example, if Employee Raj decides to close his account, Account record with Acc# 120002 has to be deleted. But this deletion is not possible as long as the Employee record of Raj references it. Hence the strategy can be to update the EmpAcc# field in the employee record of Raj to Null and then delete the Account parent record of 120002. After the deletion the data in the tables will be as follows:

**Employee**

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp#** | **EmpName** | **EmpCity** | **EmpAcc#** |
| X101 | Shekhar | Bombay | 120001 |
| X102 | Raj | Pune | ~~120002~~ Null |
| X103 | Sharma | Nagpur | Null |
| X104 | Vani | Bhopal | 120003 |

**Account**

|  |  |  |
| --- | --- | --- |
| **ACC#** | **OpenDate** | **BalAmt** |
| 120001 | 30-Aug-1998 | 5000 |
| ~~120002~~ | ~~29-Oct-1998~~ | ~~1200~~ |
| 120003 | 01-Jan-1999 | 3000 |
| 120004 | 04-Mar-1999 | 500 |

**3.4** **Relational Algebra Operators**

The eight relational algebra operators are   
  
**1. SELECT** – To retrieve specific tuples/rows from a relation.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| http://elearning.tvm.tcs.co.in/dbms/dbms/select.jpg     |  |  |  | | --- | --- | --- | | **Ord#** | **OrdDate** | **Cust#** | | 101 | 02-08-94 | 002 | | 104 | 18-09-94 | 002 | |

**2. PROJECT** – To retrieve specific attributes/columns from a relation.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| http://elearning.tvm.tcs.co.in/dbms/dbms/project.jpg   |  |  | | --- | --- | | **Descr** | **Price** | | Power Supply | 4000 | | 101-Keyboard | 2000 | | Mouse | 800 | | MS-DOS 6.0 | 5000 | | MS-Word 6.0 | 8000 | |

**3. PRODUCT** – To obtain all possible combination of tuples from two relations.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| http://elearning.tvm.tcs.co.in/dbms/dbms/product.jpg   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **Ord#** | **OrdDate** | **O.Cust#** | **C.Cust#** | **CustName** | **City** | | 101 | 02-08-94 | 002 | 001 | Shah | Bombay | | 101 | 02-08-94 | 002 | 002 | Srinivasan | Madras | | 101 | 02-08-94 | 002 | 003 | Gupta | Delhi | | 101 | 02-08-94 | 002 | 004 | Banerjee | Calcutta | | 101 | 02-08-94 | 002 | 005 | Apte | Bombay | | 102 | 11-08-94 | 003 | 001 | Shah | Bombay | | 102 | 11-08-94 | 003 | 002 | Srinivasan | Madras | |

**4. UNION** – To retrieve tuples appearing in either or both the relations participating in the UNION.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| http://elearning.tvm.tcs.co.in/dbms/dbms/union.jpg  Eg: Consider the relation Ord\_Jul as follows (Table: Ord\_Jul)   |  |  |  | | --- | --- | --- | | **Ord#** | **OrdDate** | **Cust#** | | 101 | 03-07-94 | 001 | | 102 | 27-07-94 | 003 | | 101 | 02-08-94 | 002 | | 102 | 11-08-94 | 003 | | 103 | 21-08-94 | 003 | | 104 | 28-08-94 | 002 | | 105 | 30-08-94 | 005 | |

Note: The union operation shown above logically implies retrieval of records of Orders placed in July or in August

**5. INTERSECT**- To retrieve tuples appearing in both the relations participating in the INTERSECT.

|  |
| --- |
| http://elearning.tvm.tcs.co.in/dbms/dbms/intersect.jpg  Eg:  To retrieve Cust# of Customers who've placed orders in July and in August  **Cust#**  003 |

**6. DIFFERENCE** – To retrieve tuples appearing in the first relation participating in the DIFFERENCE but not the second.

|  |
| --- |
| http://elearning.tvm.tcs.co.in/dbms/dbms/difference.jpg  Eg:  To retrieve Cust# of Customers who've placed orders in July but not in August  **Cust#**  001 |

**7. JOIN** – To retrieve combinations of tuples in two relations based on a common field in both the relations.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| http://elearning.tvm.tcs.co.in/dbms/dbms/join.jpg  Eg:  ORD\_AUG join CUSTOMERS (here, the common column is Cust#)   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Ord#** | **OrdDate** | **Cust#** | **CustNames** | **City** | | 101 | 02-08-94 | 002 | Srinivasan | Madras | | 102 | 11-08-94 | 003 | Gupta | Delhi | | 103 | 21-08-94 | 003 | Gupta | Delhi | | 104 | 28-08-94 | 002 | Srinivasan | Madras | | 105 | 30-08-94 | 005 | Apte | Bombay | |

Note: The above join operation logically implies retrieval of details of all orders and the details of the **corresponding** customers who placed the orders.

Such a join operation where only those rows having corresponding rows in the both the relations are retrieved is called the **natural join** or **inner join**. This is the most common join operation.

Consider the example of EMPLOYEE and ACCOUNT relations.

EMPLOYEE

|  |  |  |  |
| --- | --- | --- | --- |
| **EMP #** | **EmpName** | **EmpCity** | **Acc#** |
| X101 | Shekhar | Bombay | 120001 |
| X102 | Raj | Pune | 120002 |
| X103 | Sharma | Nagpur | Null |
| X104 | Vani | Bhopal | 120003 |

ACCOUNT

|  |  |  |
| --- | --- | --- |
| **Acc#** | **OpenDate** | **BalAmt** |
| 120001 | 30. Aug. 1998 | 5000 |
| 120002 | 29. Oct. 1998 | 1200 |
| 120003 | 1. Jan. 1999 | 3000 |
| 120004 | 4. Mar. 1999 | 500 |

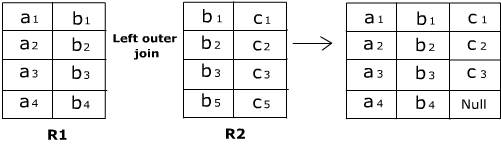
A join can be formed between the two relations based on the common column Acc#. The result of the (inner) join is :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Emp#** | **EmpName** | **EmpCity** | **Acc#** | **OpenDate** | **BalAmt** |
| X101 | Shekhar | Bombay | 120001 | 30. Aug. 1998 | 5000 |
| X102 | Raj | Pune | 120002 | 29. Oct. 1998 | 1200 |
| X104 | Vani | Bhopal | 120003 | 1. Jan 1999 | 3000 |

Note that, from each table, only those records which have corresponding records in the other table appear in the result set. This means that result of the inner join shows the details of those employees who hold an account along with the account details.

The other type of join is the outer join which has three variations – the left outer join, the right outer join and the full outer join. These three joins are explained as follows:

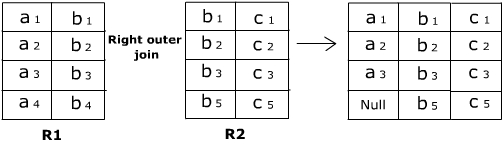
The left outer join retrieves all rows from the left-side (of the join operator) table. If there are corresponding or related rows in the right-side table, the correspondence will be shown. Otherwise, columns of the right-side table will take null values.



EMPLOYEE left outer join ACCOUNT gives:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Emp#** | **EmpName** | **EmpCity** | **Acc#** | **OpenDate** | **BalAmt** |
| X101 | Shekhar | Bombay | 120001 | 30. Aug. 1998 | 5000 |
| X102 | Raj | Pune | 120002 | 29. Oct. 1998 | 1200 |
| X103 | Sharma | Nagpur | NULL | NULL | NULL |
| X104 | Vani | Bhopal | 120003 | 1. Jan 1999 | 3000 |

The right outer join retrieves all rows from the right-side (of the join operator) table. If there are corresponding or related rows in the left-side table, the correspondence will be shown. Otherwise, columns of the left-side table will take null values.

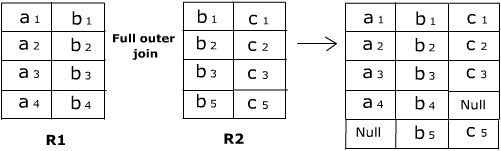


EMPLOYEE right outer join ACCOUNT gives:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Emp#** | **EmpName** | **EmpCity** | **Acc#** | **OpenDate** | **BalAmt** |
| X101 | Shekhar | Bombay | 120001 | 30. Aug. 1998 | 5000 |
| X102 | Raj | Pune | 120002 | 29. Oct. 1998 | 1200 |
| X104 | Vani | Bhopal | 120003 | 1. Jan 1999 | 3000 |
| NULL | NULL | NULL | 120004 | 4. Mar. 1999 | 500 |

(Assume that Acc# 120004 belongs to someone who is not an employee and hence the details of the Account holder are not available here)

The full outer join retrieves all rows from both the tables. If there is a correspondence or relation between rows from the tables of either side, the correspondence will be shown. Otherwise, related columns will take null values.



EMPLOYEE full outer join ACCOUNT gives:

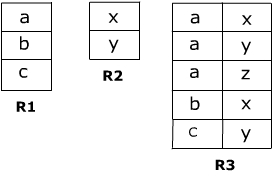
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Emp#** | **EmpName** | **EmpCity** | **Acc#** | **OpenDate** | **BalAmt** |
| X101 | Shekhar | Bombay | 120001 | 30. Aug. 1998 | 5000 |
| X102 | Raj | Pune | 120002 | 29. Oct. 1998 | 1200 |
| X103 | Sharma | Nagpur | NULL | NULL | NULL |
| X104 | Vani | Bhopal | 120003 | 1. Jan 1999 | 3000 |
| NULL | NULL | NULL | 120004 | 4. Mar. 1999 | 500 |

Q: What will the result of a natural join operation between R1 and R2 ?   
A:

|  |  |  |
| --- | --- | --- |
| a1 | b1 | c1 |
| a2 | b2 | c2 |
| a3 | b3 | c3 |

**8. DIVIDE**

Consider the following three relations:



R1 divide by R2 per R3 gives:

|  |
| --- |
| a |

Thus the result contains those values from R1 whose corresponding R2 values in R3 include all R2 values.