**UNIT-II –REALTIONAL MODEL**

**Relational Database** :-

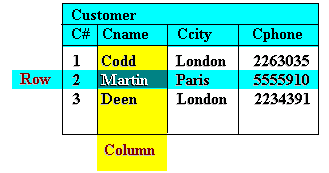
It is a database based on the relational model of data, as proposed by E. F. Codd in 1970. A software system used to maintain relational databases is a **relational database management system (RDBMS**). Virtually all relational database systems use **SQL (Structured Query Language)** for querying and maintaining the database.

## Relational model

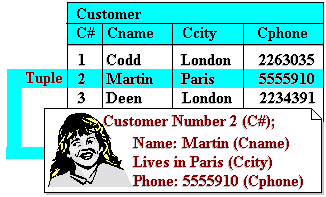
Relational model represents data in the form of relations or tables.

***Relation:-***

The term **relation** in this model refers to a **two-dimensional table** of data. In other words, according to the model, information is arranged in columns and rows. More specifically, the values in any row are not homogenous. Values in any given column, however, are all of the same type .

  
**Figure :**A Relation

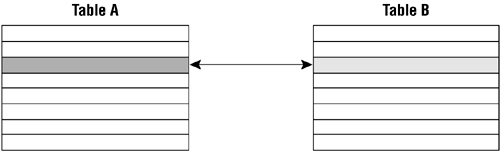
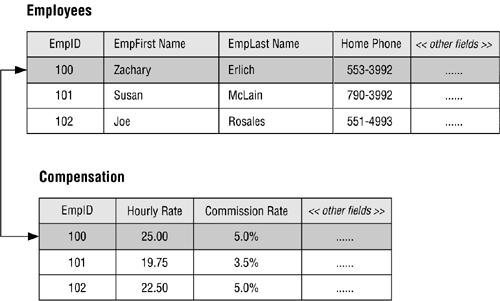
A relation has a unique name and represents a particular entity.

  
**Figure :**Relation and Entity

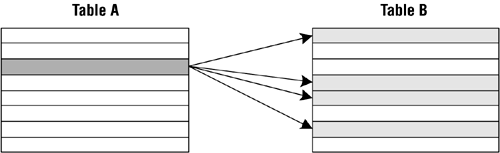
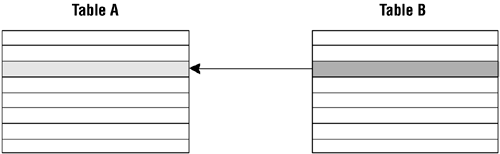
**Kinds of Relations**

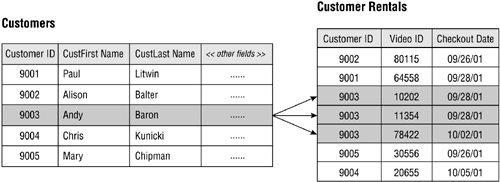
Different type of relation in RDBMS are :-

1. **One-to-one:** One to one is implemented using single table by establishing relationship between same type of columns in a table.

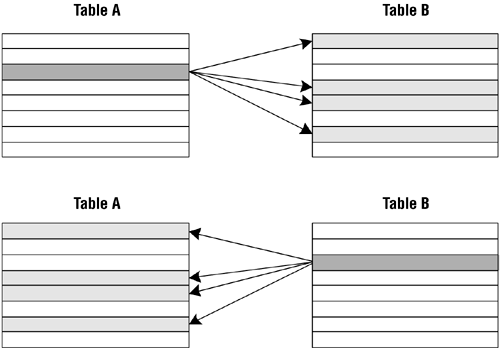
 

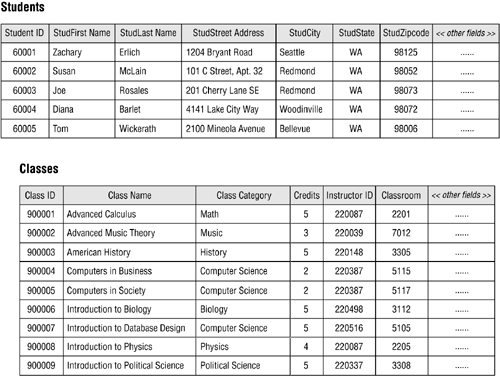
**2) One-to-many:** Implemented using two tables with primary key and foreign key relationships.

****

**3) Many-to-many:** A pair of tables bears a many-to-many relationship when a single record in the first table can be related to one or more records in the second table and a single record in the second table can be related to one or more records in the first table. Implemented using a junction table. The keys from both the tables form composite primary key of the junction table.

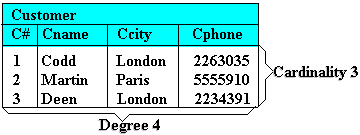




A student can attend one or more classes during a school year, so a single record in the STUDENTS table can be related to one or more records in the CLASSES table. Conversely, one or more students will attend a given class, so a single record in the CLASSES table can be related to one or more records in the STUDENTS table.

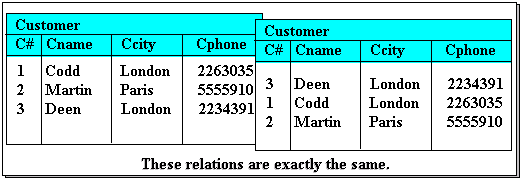
**Properties of a Relation**

A relation with **N** columns and **M** rows (tuples) is said to be of *degree* **N** and *cardinality* **M**. This is illustrated in below Figure which shows the Customer relation of degree 4 and cardinality 3. The product of a relation’s degree and cardinality is the number of attribute values it contains.

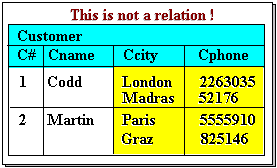
  
**Figure :**Degree and cardinality of a relation

The characteristic properties of a relation are as follows:

* All entries in a given column are of the same kind or type
* The ordering of columns is immaterial.

  
**Figure :**Column ordering is unimportant

* No two tuples are exactly the same. A relation is a set of tuples. Thus a table that contains duplicate tuples is not a relation and cannot be stored in a relational database.
* There is only one value for each attribute of a tuple. Thus a table shown in below Figure is not allowed in the relational model, despite the clear intended representation, ie. that of customers with two abodes (eg. Codd has one in London and one in Madras). In situations like this, the multiple values must be split into multiple tuples to be a valid relation.

  
**Figure :**A tuple attribute may only have one value

 The ordering of tuples is immaterial

**Tuple:** Each row of a relation is known as tuple, is a collection of facts (values) about a particular individual of that entity. In other words, a tuple represents an **instance** of the entity represented by the relation. e.g.; CUSTOMER relation has 3 tuples.

**Attribute:** Column of table represents attributes. Each relation is defined in terms of some properties known as attribute. For Example, STUD\_NO, STUD\_NAME etc. are attributes of relation STUDENT.

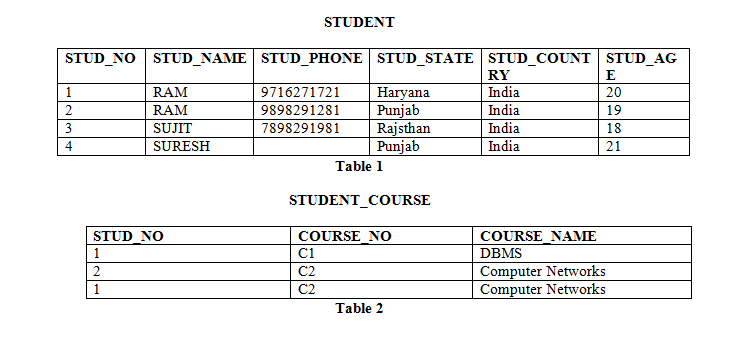
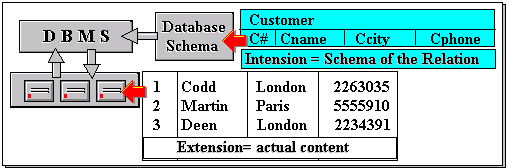


Table 1 and Table 2 represent relational model having two relations STUDENT and STUDENT\_COURSE.

**Relational Schema:**  Schema represents structure or design of a relation. e.g.; Relational Schema of STUDENT relation can be represented as:  
STUDENT (STUD\_NO, STUD\_NAME, STUD\_PHONE, STUD\_STATE, STUD\_COUNTRY, STUD\_AGE)

The more permanent parts of a relation, viz. the relation name and attribute names, are collectively referred to as its intension or schema.

  
**Figure :**The Intension and extension of a relation

**Relational Instance:** The set of values present in a relation at a particular instance of time is known as relational instance

**Domain of an attribute:** A domain is the original sets of atomic values used to model data. By atomic value, we mean that each value in the domain is indivisible as far as the relational model is concerned. For example:

* The domain of Marital Status has a set of possibilities: Married, Single, Divorced.
* The domain of Shift has the set of all possible days: {Mon, Tue, Wed…}.
* The domain of Salary is the set of all floating-point numbers greater than 0 and less than 0200,000.
* domain of STUD\_AGE can be from 18 to 40.

In short , the possible values an attribute can take in a relation is called its **domain**. It is based on various properties and the data type for the column.

**Domain Constraints**

Attributes have specific values in real-world scenario. For example, age can only be a positive integer. The same constraints have been tried to employ on the attributes of a relation. Every attribute is bound to have a specific range of values. For example, age cannot be less than zero and telephone numbers cannot contain a digit outside 0-9.

**NULL values:** Values of some attribute for some tuples may be unknown, missing or undefined which are represented by NULL. Two NULL values in a relation are considered different from each other.

**Relational Database Objects**

A database object is any defined object in a database that is used to store or reference data. Anything which we make from **create** command is known as **Database Object**. It can be used to hold and manipulate the data. Some of the examples of database objects are :

* **Table** – Basic unit of storage; composed rows and columns
* **View** – Logically represents subsets of data from one or more tables
* **Sequence** – Generates primary key values
* **Index** – Improves the performance of some queries
* **Synonym** – Alternative name for an object

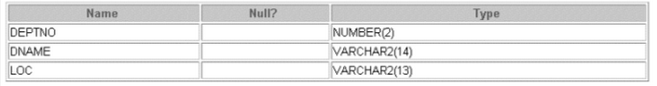
**Different database Objects :**

**1. Table** – This database object is used to create a table in database.

*Syntax* : CREATE TABLE table (column datatype [DEFAULT expr][, ...]);

*Example :* CREATE TABLE dept (deptno NUMBER(2), dname VARCHAR2(14), loc VARCHAR2(13));

*Output* : DESCRIBE dept;

 Table output

2. **View** – This database object is used to create a view in database. **A view is a logical table based on a table or another view**. Views are **virtual table**s. When we fire query on these views, the underlying view query will be executed. Hence it does not consume any space in memory to store the data nor does it create same copies of data in the database. It simply shows the records from the table itself.

A table will have large number of data and table will be fired with specific frequently. In such case, instead of rewriting the query again and again, a name is given to the query and it will be called whenever it is required. Hence view is also called as **Named Query or Stored Query.**

The view is stored as a SELECT statement in the data dictionary.

A view can be **created** as follows :

*Syntax :*CREATE OR REPLACE VIEW view\_name AS

SELECT COLUMN1, COLUMN2 …COLUMNN FROM table\_name

WHERE [Condition];

*Example :*

CREATE [OR REPLACE] VIEW vw\_RedPen AS SELECT \* FROM Pen WHERE INK\_COLOR = ‘Red’;

* Example of view with more than one table :

CREATE OR REPLACE VIEW vw\_Design\_Emp AS

SELECT e.EMP\_ID, e.EMP\_FIRST\_NAME, e.EMP\_LAST\_NAME, d.DEPT\_ID, d.DEPT\_NAME FROM EMPLOYEE e, DEPARTMENT d –

WHERE e.DEPT\_ID = d.DEPT\_ID -- Join condition

AND d.DEPT\_NAME = ‘DESIGN’; -- Filtering condition

A view can be queried same way a table is queried – by using SELECT. We can even have conditions, sorting, grouping, relation operator etc like we do in tables.

*Output :-* SELECT \* FROM vw\_RedPen;

SELECT \* FROM vw\_Design\_Emp WHERE EMP\_ID = 12121;

We can **DROP** the view by writing:

DROP VIEW view\_name;

DROP VIEW vw\_RedPen;

DROP VIEW vw\_Design\_Emp;

**Base Tables** :- The tables on which a view is based are called **BASE TABLES**.

In above Examples Base tables are :- Pen,Employee, Department

**3. Sequence** – This database object is used to create a sequence in database. A typical usage for sequences is to create a **primary key value**, which must be unique for each row. The sequence is generated and incremented (or decremented) by an internal Oracle routine.

*Syntax :* CREATE SEQUENCE sequence [INCREMENT BY n] [START WITH n]

[{MAXVALUE n | NOMAXVALUE}] [{MINVALUE n | NOMINVALUE}];

//[{CYCLE | NOCYCLE}] [{CACHE n | NOCACHE}];

*Example :* CREATE SEQUENCE dept\_deptid\_seq INCREMENT BY 10 START WITH 120

MAXVALUE 9999

// NOCACHE NOCYCLE;

**4. Index** – This database object is used to create a indexes in database which help in speeding up the retrieval of rows by using a pointer. Indexes are logically and physically independent of the table they index. This means that they can be **created or dropped** at any time and have no effect on the base tables or other indexes.

*Syntax :* CREATE INDEX index ON table (column[, column]...);

*Example :* CREATE INDEX emp\_last\_name\_idx ON employees(last\_name);

**5. Synonym** – This database object simplify access to objects by creating a synonym (another name for an object). With synonyms, we can Ease referring to a table owned by another user and shorten lengthy object names.

In the syntax:

PUBLIC : creates a synonym accessible to all users

synonym : is the name of the synonym to be created

object : identifies the object for which the synonym is created

*Syntax :* CREATE [PUBLIC] SYNONYM synonym FOR object;

*Example :* CREATE SYNONYM d\_sum FOR dept\_sum\_vu;

**CATALOG**

The catalog consists of metadata in which definitions regarding database objects like base tables, views, synonyms,  indexes, users, and user groups are stored. In computing, a catalog is a directory of information about data sets, files, or a database . A catalog usually describes where a data set, file or database entity is located and may also include other information, such as the type of device on which each data set or file is stored.

## Metadata0

## �      Information on

### �    Tables in the system

### �    Attributes in tables

### �    Primary keys

### �    Secondary keys

### �    Foreign keys

### �    Information regarding indexes

### �    Definitions of views

### �    Security information(who gets to see what information)

# The catalog is stored as relations

## �      The catalog can be queried using SQL statements for metadata

**Data Dictionary** contains meta data information of each and every schemas object.

**Relation and Predicates**

A predicate is an SQL expression that evaluates a search condition that is either TRUE, FALSE or UNKNOWN. .Predicates are specified in the WHERE clause of a SQL statement.

In the example below "EMPNO=7788" is a predicate:

SELECT \* FROM emp WHERE **empno = 7788[predicate];**

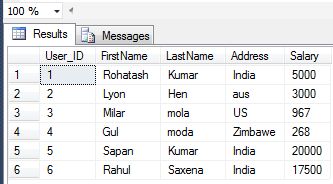
Various types of Predictaes are as follow:

* comparison (=, <>, <, >, <=, >=, **!=**)
* BETWEEN
* LIKE
* EXITS | NOT EXISTS
* IN | NOT IN
* NULL

**COMPARISON**

The COMPARISON predicates compare two values. If either value is NULL, then the result of the predicate is UNKNOWN.

|  |  |  |
| --- | --- | --- |
| ***Table 1 : Comparison Predicate Symbols*** | | |
| **Comparison Symbol** | **Symbol Description** | **Result**  **Description** |
| = | equal to | This symbol results to TRUE if both values are the same. |
| <> or != | not equal to | This symbol results to TRUE if the first value is equal to the second value. |
| < | less than | This symbol results to TRUE if the first value is less than the second value. |
| > | greater than | This symbol results to TRUE if the first value is greater than the second value. |
| <= | less than or equal to | This symbol results to TRUE if the first value is less than or equal to the second value. |
| >= | greater than or equal to | This symbol results to TRUE if the first value is greater than or equal to the second value. |

****

**Examples**

The following are examples of using the comparison predicates. The results (TRUE, FALSE, or UNKNOWN) of the predicates are based on the values of the column.

* User\_id = 2 ---> TRUE if User\_id is 2
* Address != ‘Paris' ----> TRUE if the Address is not PARIS
* Salary > 18000 ----> TRUE if salary is greater than $18000

**BETWEEN**

The BETWEEN predicate determines if a value is between a range of values.

// This operator is used with AND operator. The Between operator can be used  with numeric, text and date data types.

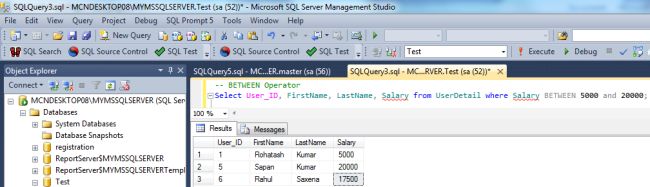
expression [NOT] BETWEEN literal AND literal

**Example**

Select User\_ID, FirstName, LastName, Salary from UserDetail

where Salary BETWEEN 5000 and 20000;

**Output**



**LIKE Operator**

LIKE is an operator that compares column values with a specified pattern. During pattern matching, regular characters must exactly match the characters specified in the character string.

/\* The data type of the column can be any character or date data type. There are certain characters within the pattern, called wildcard characters.  I have used four types of wildcards; they are:

1. Percent sign (%): It is used to represent or search any string of **zero or more characters.**
2. Underscore (\_): It is used to represent or search **a single character.**
3. Bracket ([]): It is used to represent or search any **single character within the specified range.**
4. Caret (^): It is used to represent or search any single character **not** within the specified range. \*/

**Example**

Select User\_ID, FirstName, LastName, Salary from UserDetail

 where FirstName LIKE '%h%';

**Output**



#### IN | NOT IN

You can use these predicate keywords to return a value list or a subquery.

SELECT FirstName, LastName, City, Salary WHERE Salary IN (5000,20000);

**Output**

Rohatash Kumar India

Sapan Kumar India

#### NULL

The NULL predicate determines if a column in a selected row contains the SQL value: NULL. If the column value is NULL, returns TRUE. The following is the syntax for the NULL predicate:

column\_name IS [NOT] NULL

**Examples**

In the first example, the NULL predicate looks for any row where the column contains a NULL value. In the second example, the NULL predicate looks for all rows that do not contain a NULL value for a column.

* 1. Select \* from UserDeatil where FirstName IS NULL
  2. Select \* from UserDeatil where FirstName IS NOT NULL

**keys in DBMS**

Key plays an important role in relational database; it is used for identifying unique rows from table. It also establishes relationship among tables.

### Types of keys in DBMS

* Primary Key – A primary is a column or set of columns in a table that uniquely identifies tuples (rows) in that table.
* Super Key – A super key is a set of one of more columns (attributes) to uniquely identify rows in a table.
* Candidate Key – A super key with no redundant attribute is known as candidate key.
* Alternate Key – Out of all candidate keys, only one gets selected as primary key, remaining keys are known as alternate or secondary keys.
* Composite Key – A key that consists of more than one attribute to uniquely identify rows (also known as records & tuples) in a table is called composite key.
* Foreign Key – Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables.

# Primary key in DBMS

**Definition**: A primary is a column or set of columns in a table that uniquely identifies tuples (rows) in that table.

**Example**:  
Student Table

|  |  |  |
| --- | --- | --- |
| Stu\_Id | Stu\_Name | Stu\_Age |
| 101 | Steve | 23 |
| 102 | John | 24 |
| 103 | Robert | 28 |
| 104 | Carl | 22 |

In the above Student table, the Stu\_Id column uniquely identifies each row of the table.

**Note**:

* We denote the primary key by **underlining** the column name.
* The value of primary key should be unique for each row of the table. Primary key column **cannot contain duplicate values**.
* Primary key column should **not contain nulls**.
* Primary keys are not necessarily to be a single column; more than one column can also be a primary key for a table. For e.g. {Stu\_Id, Stu\_Name} collectively can play a role of primary key in the above table, but that does not make sense because Stu\_Id alone is enough to uniquely identifies rows in a table then why to  make things complex. Having that said, we should choose more than one columns as primary key only when there is no single column that can play the role of primary key.

**How to choose a primary key?**

There are two ways: **Either** to create a column and let database automatically have numbers in increasing order for each row **or** choose a column yourself making sure that it does not contain duplicates and nulls. For e.g. in the above Student table, The Stu\_Name column cannot be a primary key as more than one people can have same name, similarly the Stu\_Age column cannot play a primary key role as more than one persons can have same age.

# 2. Super key in DBMS

**Definition**: A super key is a set or one of more columns (attributes) to uniquely identify rows in a table. Often people get confused between super key and candidate key, so we will also discuss a little about candidate key here.

**How candidate key is different from super key?**

Answer is simple – Candidate keys are selected from the set of super keys, the only thing we take care while selecting candidate key is*: It should not have any redundant attribut*e. That’s the reason they are also termed as **minimal super key.**

Let’s take an example to understand this: **Employee table**

|  |  |  |
| --- | --- | --- |
| Emp\_SSN | Emp\_Number | Emp\_Name |
| 123456789 | 226 | Steve |
| 999999321 | 227 | Ajeet |
| 888997212 | 228 | Chaitanya |
| 777778888 | 229 | Robert |

**Super keys**:

* {Emp\_SSN}
* {Emp\_Number}
* {Emp\_SSN, Emp\_Number}
* {Emp\_SSN, Emp\_Name}
* {Emp\_SSN, Emp\_Number, Emp\_Name}
* {Emp\_Number, Emp\_Name}

All of the above sets are able to uniquely identify rows of the employee table.

**Candidate Keys**:

As stated above, they are the **minimal super keys with no redundant attributes.**

* {Emp\_SSN}
* {Emp\_Number}

Only these two sets are candidate keys as all other sets are having redundant attributes that are not necessary for unique identification.

## Super key vs Candidate Key

Clear explanation:

1. First we have to understand that **all the candidate keys are super keys**. This is because the candidate keys are chosen out of the super keys.
2. How we choose candidate keys from the set of super keys?

We look for those keys from which we cannot remove any fields. In the above example, we have not chosen {Emp\_SSN, Emp\_Name} as candidate key because {Emp\_SSN} alone can identify a unique row in the table and Emp\_Name is redundant.

**Primary key**:

Primary key is being selected from the sets of candidate keys by database designer. So Either {Emp\_SSN} or {Emp\_Number} can be the primary key.

# Foreign key in DBMS

**Definition**: Foreign keys are the columns of a table that points to the primary key of another table. They act as a cross-reference between tables.

**For example**:

In the below example the **Stu\_Id** column in **Course\_enrollment** table is a foreign key as it points to the primary key of the Student table.

**Course\_enrollment table:**

|  |  |
| --- | --- |
| Course\_Id | Stu\_Id |
| C01 | 101 |
| C02 | 102 |
| C03 | 101 |
| C05 | 102 |
| C06 | 103 |
| C07 | 102 |

**Student table:**

|  |  |  |
| --- | --- | --- |
| Stu\_Id | Stu\_Name | Stu\_Age |
| 101 | Chaitanya | 22 |
| 102 | Arya | 26 |
| 103 | Bran | 25 |
| 104 | Jon | 21 |

**Note**: Practically, the foreign key has nothing to do with the primary key tag of another table, if it points to a unique column (not necessarily a primary key) of another table then too, it would be a foreign key. So, a correct definition of foreign key would be: Foreign keys are the columns of a table that points to the candidate key of another table.

# Candidate Key in DBMS

A super key with no redundant attribute is known as candidate key. Candidate keys are selected from the set of super keys, the only thing we take care while selecting candidate key is: It should not have any redundant attributes. That’s the reason they are also termed as minimal super key.

**For example**:

|  |  |  |
| --- | --- | --- |
| Emp\_Id | Emp\_Number | Emp\_Name |
| E01 | 2264 | Steve |
| E22 | 2278 | Ajeet |
| E23 | 2288 | Chaitanya |
| E45 | 2290 | Robert |

There are two candidate keys in above table:

{Emp\_Id}  
{Emp\_Number}

**Note**: A primary key is being selected from the group of candidate keys. That means we can either have Emp\_Id or Emp\_Number as primary key.

# Composite key in DBMS

A key that consists of more than one attribute to uniquely identify rows (also known as records & tuples) in a table is called composite key. It is also known as compound key.

**Example: Table – Sales**

|  |  |  |  |
| --- | --- | --- | --- |
| cust\_Id | order\_Id | product\_code | product\_count |
| C01 | O001 | P007 | 23 |
| C02 | O123 | P007 | 19 |
| C02 | O123 | P230 | 82 |
| C01 | O001 | P890 | 42 |

Key in above table: {cust\_id, order\_id}

This is a composite key as it consists of more than one attribute.

# Alternate key in DBMS

Out of all [candidate keys](https://beginnersbook.com/2015/04/candidate-key-in-dbms/), only one gets selected as [primary key](https://beginnersbook.com/2015/04/primary-key-in-dbms/), remaining keys are known as alternative or secondary keys.

**For example: Consider the below table**

|  |  |  |
| --- | --- | --- |
| Emp\_Id | Emp\_Number | Emp\_Name |
| E01 | 2264 | Steve |
| E22 | 2278 | Ajeet |
| E23 | 2288 | Chaitanya |
| E45 | 2290 | Robert |

There are two candidate keys in above table:

{Emp\_Id}  
{Emp\_Number}

Since we have selected Emp\_Id as primary key, the remaining key Emp\_Number would be called alternative or secondary key.

**Weak and Strong Entity Set**

An entity set that does not possess sufficient attributes to form a primary key is called a **weak entity set.** One that does have a primary key is called a **strong entity set**.

For example,

* The entity set *transaction* has attributes *transaction-number*, *date* and *amount*.
* Different transactions on different accounts could share the same number.
* These are not sufficient to form a primary key (uniquely identify a transaction).
* Thus *transaction* is a weak entity set.

For a weak entity set to be meaningful, it must be part of a one-to-many relationship set. This relationship set should have no descriptive attributes. (Why?)

The idea of strong and weak entity sets is related to the existence dependencies seen earlier.

* Member of a strong entity set is a dominant entity.
* Member of a weak entity set is a subordinate entity.

A weak entity set does not have a primary key, but we need a means of distinguishing among the entities.

The **discriminator** of a weak entity set is a set of attributes that allows this distinction to be made.

The **primary key of a weak entity set** is formed by taking the primary key of the strong entity set on which its existence depends (see Mapping Constraints) plus its **discriminator**.

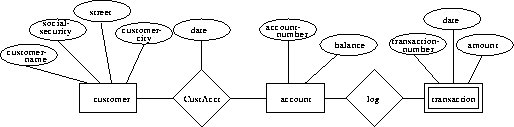
To illustrate:

* *transaction* is a weak entity. It is existence-dependent on *account*.
* The primary key of *account* is *account-number*.
* *transaction-number* distinguishes transaction entities within the same account (and is thus the discriminator).
* So the primary key for *transaction* would be *(account-number, transaction-number)*.

**Just Remember:** The primary key of a weak entity is found by taking the primary key of the strong entity on which it is existence-dependent, plus the discriminator of the weak entity set.

**Reducing E-R Diagrams to Tables**

A database conforming to an E-R diagram can be represented by a collection of tables. We'll use the E-R diagram of Figure [2.14](https://www.cs.sfu.ca/CourseCentral/354/zaiane/material/notes/Chapter2/node10.html#fig213strong)) as our example.

     
**Figure 2.14:** E-R diagram with strong and weak entity sets