Journal A – Database Systems Omar Salih

1a. Determinant (note this is the most important concept in the course). Provide two illustrations of the concept with real word data attributes (not X, Y etc.). In examples list the determinant and the dependents as well.

**Definition:**A determinant in a database table is any attribute that you can use to determine the values assigned to other attribute(s) in the same row.

**Examples:**Consider a table with the attributes employee\_id, first\_name, last\_name and date\_of\_birth. In this case, the field employee\_id determines the remaining three fields. The name fields do not determine the employee\_id because the firm may have more than one employee with the same first and/or last name. Similarly, the DOB field does not determine the employee\_id or the name fields because more than one employee may share the same birthday.

Example: Consider this (obviously non-normal) table:

CREATE TABLE US\_Address (

AddressID int,

Streetline varchar(80),

City varchar(80),

State char(2),

ZIP char(5),

StateName varchar(80),

StateTax DECIMAL(5,2)

)

State is a determinant for StateName and StateTax, but it is not a candidate key for the row. Proper normalization, would therefore move StateName and StateTax out of the US\_Address table and into a States table.

------

Even if a table is in BCNF, every subset of attributes is a determinant. What is correct is "A [non-trivial FD's] determinant may not be [a superset of] a candidate key if the table is not [in BCNF] [but otherwise is]"

Also, that link unusually defines "determinant" (in a table) as "determinant of a full functional dependency". And its "A relation is in BCNF if, and only if, every determinant [sic] is a candidate key" should be "every non-trivial determinant [sic]"

-------

* A primary key or any candidate key is also a determinant while the opposite is not true.
* A determinant can uniquely determine one or more attributes in the row.
* A candidate key can uniquely determine the entire row.

Taking an example from [here](http://www.sitepoint.com/database-design-management/), let there be a table with following columns:

Customer #, Name, Address, Credit, Sales Rep #, Sales Rep Name

and let's say that the Sales Rep # can uniquely determine the Sales Rep Name. Thus, Sales Rep #is a determinant for Sales Rep Name but is not a candidate key for this table.

------

"**determinant**" and "**candidate key**" are not the same concept. Note that a determinant can be of an FD or of a table. A CK can only be of a table. Every CK is a determinant of its table.

For sets of columns X and Y we can write *X -> Y*. **We say that X is the *determinant set* and Y is the *determined set* of/in *functional dependency* (*FD*) X -> Y.**

We say X *functionally determines* Y and Y *is functionally determined by* X. We say X is *the determinant* of X -> Y. In {C} -> Y we say C *functionally determines* Y. In X -> {C} we say X *functionally determines* C. When X is a superset of Y we say X -> Y is *trivial*.

**We say X -> Y *holds in* table T when each subrow of values for X always/only appears with the same subrow of values for Y.** Or we say X -> Y is a FD *of/in* T. **When X is a determinant of some FD in table T we say X *is a determinant of/in* T.**

**A *superkey* of a table T is a set of columns that functionally determines every attribute. A*candidate key* (*CK*) is a superkey that contains no smaller superkey.** We can pick one CK as*primary key* (*PK*) and then call the other CKs *alternate keys*. A column is *prime* when it is in some CK.

Note that a determinant can be *of a FD* or *of (a FD that holds in) a table*. **Every CK is a determinant of its table.** (But then, in a table *every* set of columns is a determinant: of itself, trivially. And similarly*every* column.)

(These definitions do not depend on normalization. FDs and CKs of a table are used in normalizing it. A table is in BCNF when every determinant *of a non-trivial FD* that holds in it is a *superkey*.)

----

|  |  |  |
| --- | --- | --- |
| |  | | --- | |  | | The definitions do not *depend* on normalization, but I'll wager they were *invented* explicitly to make it possible to define normalization. Your definitions are very precise and mathematical. But while they carry very valuable information, reading them is much harder than my answer, incorrect though it was. My answer quickly got the OP to understand that they were different concepts and where the different concepts would be useful. Your answer is more about formal proof. – [Ross Presser](http://stackoverflow.com/users/864696/ross-presser) [Dec 10 '15 at 17:08](http://stackoverflow.com/questions/16706637/are-determinants-and-candidate-keys-same-or-different-things#comment56159923_34191604) |
| |  |  | | --- | --- | |  |  | | The *terms* appear in the question and your answer but the *meanings* aren't in the asker's mind. I tried to address that. Hence definitions. They are simple and correct yet not vague. The four boldface sentences for*FD*, *holds*, *superkey* and *CK* would have sufficed. I added some related terms/notions. My choice reflects my doubt that difference can be appreciated between things without clear understanding of just what they are. – [philipxy](http://stackoverflow.com/users/3404097/philipxy" \o "6,106 reputation) [Dec 10 '15 at 22:55](http://stackoverflow.com/questions/16706637/are-determinants-and-candidate-keys-same-or-different-things#comment56170974_34191604) |

1b. Difference between Super key and Primary Key. Provide an example of each.

Minimal column which are sufficient to identify row is **primary key**. **Super key** also use for identify row but one **super key** may be contain more than 1 **primary key** or combination of **primary keys** known as **super key**. Both used for uniquely identify of row. Table contain more than 1 candidate **key** but only 1 **primary key**.

-------

A Super Key is simply a non-minimal Candidate Key, that is to say one with additional columns not strictly required to ensure uniqueness of the row.

A Primary Key is a minimal Candidate Key, which is to say all constituent columns are strictly required in order to ensure uniqueness.

As a database developer/designer of 30 years experience, I had never even heard the term Super Keyuntil I saw this question, and looked it up. The concept of Super Key seems more germane to the topic of performance and Physical Schema design as it directly maps to the concept of a unique nonclustered index with additional columns for improved query covering.

-----

Super Keys : Super key stands for superset of a key. A Super Key is a set of one or more attributes that are taken collectively and can identify all other attributes uniquely.

For Example, We are having table

Book (BookId, BookName, Author)

So in this table we can have

¬ (BookId)

¬ (BookId,BookName)

¬ (BookId, BookName, Author)

¬ (BookId, Author)

¬ (BookName, Author)

As our Super key. Each super key is able to uniquely identify each tuple (record).

Candidate Keys Candidate keys are a super key which are not having any redundant attributes. In other words candidate keys are minimal super keys. For Example, In above illustration

¬ (BookId)

¬ (BookName,Author)

These two keys can be candidate keys, as remaining keys are having redundant attributes. Means in super key (BookId, BookName) record can be uniquely identify by just bookid and therefore Bookname is redundant attribute

Primary Key: It is a candidate key that is chosen by the database designer to identify entities with in an entity set. OR A key which is used to uniquely identify each record is known as primary key.

From above Candidate keys any one can be the primary key. And the another one which is not chosen as primary key will be know as Alternate key

p----

**Super Key:** a super key is that when you add other fields of a table to a Primary Key, like UserID + UserName is a Super Key.

**Primary Key** is a key which maintains the uniqueness in a table at row level

-----

Super key is set or one of more columns to uniquely identify rows in a table.

Candidate keys are selected from set of super keys.

Primary keys are selected from the set of candidate keys.

------

1. Minimal column which are sufficient to identify row is primary key. Super key also use for identify row but one super key may be contain more than 1 primary key or combination of primary keys known as super key. Both used for uniquely identify of row. Table contain more than 1 candidate key but only 1 primary key.
2. Yes
3. Primary key is subset of super key but vice versa not true.

----

* **Super Key** - is a set of one or more columns that can be used to identify a record uniquely in a table
* **Candidate Key** – can be any column or a combination of columns that can qualify as a unique key in database. There can be multiple Candidate Keys in one table. Each Candidate Key can qualify as a Primary Key. You can think of this as the "shortest" super key or **minimal** super key
* **Primary Key** – is a column or a combination of columns that uniquely identify a record. Only one Candidate Key can be Primary Key.

For a Candidate Key to qualify as a Primary Key, it should be unique and non-null.

So, basically a primary key is just one of the candidate keys, which is a just a minimal super key.

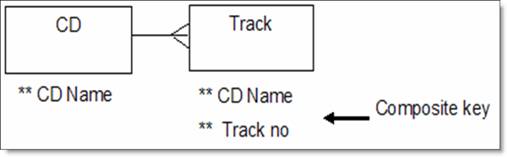
----

Informally, a **superkey** is a set of attributes within a table whose values can be used to uniquely identify a tuple. A candidate **key** is a minimal set of attributes necessary to identify a tuple; this is also called a minimal **superkey**.

A **primary key**, also called a **primary** keyword, is a **key** in a relational database that is unique for each record. It is a unique identifier, such as a driver license number, telephone number (including area code), or vehicle identification number (VIN). A relational database must always have one and only one **primary key**.

1c. Composite key. Provide an example.

A **composite key** consists of more than one attribute to uniquely identify an entity occurrence. This differs from a**compound key** in that one or more of the attributes, which make up the **key**, are not simple **keys** in their own right. For example, you have a database holding your CD collection. One of the entities is called tracks, which holds details of the tracks on a CD. This has a composite key of CD name, track number.



---

A composite key, in the context of relational databases, is a combination of two or more columns in a table that can be used to uniquely identify each row in the table. Uniqueness is only guaranteed when the columns are combined; when taken individually the columns do not guarantee uniqueness.

--

A composite key can be defined as the primary key. This is done using SQL statements at the time of table creation. It means that data in the entire table is defined and indexed on the set of columns defined as the primary key.

---

When a primary key is created from a combination of 2 or more columns, the primary key is called a composite key. Each column may not be unique by itself within the database table but when combined with the other column(s) in the composite key, the combination is unique.  
  
To illustrate the concept of the composite key and the foreign key, consider the sample table design below:  
  
customer table

|  |  |
| --- | --- |
| **column** |  |
| lastname | primary key |
| firstname | primary key |
| dateofbirth |  |

The lastname column and the firstname column together form a composite key. Let's assume that the above tables contain the following data:

|  |  |  |
| --- | --- | --- |
| **lastname** | **firstname** | **dateofbirth** |
| henry | john | 03/05/1960 |
| henry | adam | 06/08/1974 |
| kidman | adam | 04/01/1955 |
| bailey | harry | 05/05/1980 |
| morgan | alex | 09/09/1975 |

Notice that in the lastname column, there are 2 records with the value 'henry' and in the firstname column, there are 2 records with the value 'adam'. However, there are no records in the database table with a duplicate combination of both the lastname and the firstname.  
  
BTW, the above table design is very problematic, sooner or later you will have customers changing their names or customers with exactly the same name...

--

1d. Referential Integrity.

Referential integrity is a database concept that ensures that relationships between tables remain consistent. When one table has a [foreign key](http://databases.about.com/cs/specificproducts/g/foreignkey.htm) to another table, the concept of referential integrity states that you may not add a record to the table that contains the foreign key unless there is a corresponding record in the linked table. It also includes the techniques known as cascading update and cascading delete, which ensure that changes made to the linked table are reflected in the primary table.  
  
Consider the situation where we have two tables: Employees and Managers. The Employees table has a foreign key attribute entitled ManagedBy which points to the record for that employee’s manager in the Managers table. Referential integrity enforces the following three rules:

1. We may not add a record to the Employees table unless the ManagedBy attribute points to a valid record in the Managers table.
2. If the [primary key](http://databases.about.com/cs/administration/g/primarykey.htm) for a record in the Managers table changes, all corresponding records in the Employees table must be modified using a cascading update.

CONTINUE READING BELOW OUR VIDEO

Choosing The Best Smartphone For YOU

1. If a record in the Managers table is deleted, all corresponding records in the Employees table must be deleted using a cascading delete.

---

Referential integrity means that the foreign key in any referencing table must always refer to a valid row in the referenced table. Referential integrity ensures that the relationship between two tables remains synchronized during updates and deletes.

For example, assume that your application has both a Titles table and a Publishers table as shown in the following table.

|  |  |
| --- | --- |
| **Titles table** | **Publishers table** |
| ti\_isbn (key) | pu\_id (key) |
| ti\_title | pu\_name |
| ti\_yearpublished | pu\_address |
| pu\_id (foreign key) | pu\_phone |

Referential integrity requires that these two tables must be synchronized. That is, each publisher identification (pu\_id) in the Titles table must also exist in the Publishers table.

Your application cannot just delete the pu\_id row from the Publishers table because that would leave the pu\_id in the Titles table without a reference. It would be permissible, however, to delete the pu\_id row from the Publishers table and also delete every row in the Titles table that has the same pu\_id. Such an action would maintain referential integrity for these two tables.

In a similar manner, your application cannot just add a row to the Titles table without a valid pu\_id already in the Publishers table. To do so would insert "bad" data in the pu\_id field. So, your application must ensure a valid pu\_id key in the Publishers table before inserting the pu\_id in the related Titles row.

The actual implementation of referential integrity depends entirely on the data storage engine you choose and your application's design requirements. Historically, applications using mainframe VSAM files used application code to handle referential integrity. Today, even if your application uses SQL Server, that does not mean you must use triggers, foreign keys, constraints, and cascading deletes to maintain referential integrity. You might again choose to handle referential issues with application-based code.

---

**Referential integrity** is a relational database concept, which states that table relationships must always be consistent. In other words, any foreign key field must agree with the primary key that is referenced by the foreign key.

----

**Definition - What does *Referential Integrity (RI)* mean?**

Referential integrity is a relational database concept, which states that table relationships must always be consistent. In other words, any foreign key field must agree with the primary key that is referenced by the foreign key. Thus, any primary key field changes must be applied to all foreign keys, or not at all. The same restriction also applies to foreign keys in that any updates (but not necessarily deletions) must be propagated to the primary parent key.

-----

Referential Integrity is set of constraints applied to foreign key which prevents entering a row in child table (where you have foreign key) for which you don't have any corresponding row in parent table i.e. entering NULL or invalid foreign keys. *Referential Integrity* prevents your table from having  incorrect or incomplete relationship e.g. If you have two tables Order and Customer where Customer is parent table with [primary key](http://java67.blogspot.sg/2012/10/difference-between-primary-key-vs-candidate-keys-sql-database.html)customer\_id and Order is child table with foreign key customer\_id. Since as per business rules you can not have an Order without a Customer and this business rule can be implemented using *referential integrity in SQL* on relational database. **Referential Integrity** will cause failure on any INSERT or UPDATE SQL statement changing value of customer\_id in child table, If value of customer\_id is not present in Customertable. By the way What is Referential Integrity in SQL is also an important SQL question similar to [finding second highest salary in SQL](http://javarevisited.blogspot.sg/2012/12/how-to-find-second-highest-or-maximum-salary-sql.html) or [difference between truncate and delete](http://javarevisited.blogspot.sg/2011/10/how-to-use-truncate-and-delete-command.html) and should be prepared well before going for any job interview, where knowledge of SQL is one of the requirement.

Referential Integrity example in MySQL tables:

[What is Refrential Integrity in SQL database with MySQL example](http://1.bp.blogspot.com/-ALTiC41IG4I/TZ_n2bwh6iI/AAAAAAAAAFE/Jy4cmcZqUG4/s1600/mysql.gif)

Another example of Referential Integrity is Employee and Department relationship. If we have dept\_id as foreign key in Employee table than by using referential integrity constraints we can avoid creating  Employee without department or non existing department. In short Referential Integrity makes primary key foreign key relationship viable. Let's first create Employee and Department table with [primary key](http://java67.blogspot.sg/2012/10/difference-between-primary-vs-unique-key-table-sql.html), foreign key and referential Integrity constraints.

**CREATE** **TABLE** Department (dept\_id INT **NOT** **NULL**,  
                         dept\_name VARCHAR(256),                         
                         **PRIMARY** **KEY** (dept\_id)) ENGINE=INNODB;  
  
**CREATE** **TABLE** Employee (emp\_id INT **NOT** **NULL**,  
                       emp\_name VARCHAR(256),  
                       dept\_id INT,  
                       **FOREIGN** **KEY** (dept\_id) **REFERENCES** Department(dept\_id)  
                       **ON** **DELETE** CASCADE) ENGINE=INNODB;

Above SQL statements will create both Department and Employee table. dept\_id is now foreign key in Employee table. In this SQL, while creating foreign key we have specified ON DELETE clause which tells, what needs to done when a record from parent table is deleted. CASCADE referential action allows to **delete or update all matching rows from child table**, after deleting a record in parent table. This way Refrential Integrity preserve data integrity of relationship.

Let's see How Referential Integrity disallow INSERT and UPDATE for a record in child table for which there is no matching record in parent table. To check this Refrential Integrity example execute following MySQL queries :

**INSERT** **INTO** Department **VALUES** (1, "Sales");  
**INSERT** **INTO** Employee **VALUES** (101, "Rajeev", 2)  
  
mysql> **INSERT** **INTO** Employee **VALUES** (101, "Rajeev", 2)  
    -> ;  
ERROR 1452 (23000): Cannot **ADD** **OR** **UPDATE** a child row: a **FOREIGN** **KEY** constraint fails (`test`.`employee`, CONSTRAINT `employee\_ibfk\_1` **FOREIGN** **KEY** (`dept\_id`) **REFERENCES** `department` (`dept\_id`) **ON** **DELETE** CASCADE)

When we inserted first record in Department table it ran fine but when we insert a record in Employee table with dept\_id = 2 which is **not present in Department i.e. parent table,** failed to Referential integrity or foreign key constraint check.

If you modify your query and correct dept\_id to 1, query will run fine, as shown below

mysql> **INSERT** **INTO** Employee **VALUES** (101, "Rajeev", 1);  
Query OK, 1 row affected (0.05 sec)

Now let's delete our only record from Department table and see if matching records on child table is automatically deleted or not.

mysql> **DELETE** **FROM** Department;  
Query OK, 1 row affected (0.05 sec)  
  
mysql> **SELECT** \* **FROM** Employee;  
Empty **SET** (0.00 sec)

You see there is no record in Employee table because of ON DELETE CASCADE, matching records in child table is delete. Similarly you can use ON UPDATE CASCADE to automatically propagate UPDATE from parent table to child tables.

Advantage of Referential Integrity in RDBMS and SQL

There are several benefit of Referential Integrity in relational database and maintaining integrity of data among parent and child tables. Here are some of the most noticed advantages of Referential Integrity in SQL:

1) Referential Integrity prevents inserting records with incorrect details in table. Any insert or update operation will fail if it doesn't satisfy referential integrity rule.

2) If a records from parent table is deleted, referential integrity allows to delete all related records from child table using cascade-delete functionality.

3) Similar to second advantage if a record i.e. customer\_id of a customer is updated on parent table (Customer table) , Referential Integrity helps to update customer\_id in child table (Order) using cascade-update.

That's all on **What is referential integrity in database, SQL and especially in MySQL**. We have seen example of How referential integrity or foreign key constraint works in MySQL. We have also seen example of CASCADE DELETE which automatically delete matching records form child table.

Read more: <http://javarevisited.blogspot.com/2012/12/what-is-referential-integrity-in-database-sql-mysql-example-tutorial.html#ixzz4M8jrizL5>

----

A feature provided by [relational database management systems (RDBMS's)](http://www.webopedia.com/TERM/R/RDBMS.html) that prevents users or applications from entering inconsistent data. Most RDBMS's have various referential integrity rules that you can apply when you create a relationship between two [tables](http://www.webopedia.com/TERM/T/table.html).

For example, suppose Table B has a [*foreign key*](http://www.webopedia.com/TERM/K/key.html) that points to a field in Table A. Referential integrity would prevent you from adding a [record](http://www.webopedia.com/TERM/R/record.html) to Table B that cannot be linked to Table A. In addition, the referential integrity rules might also specify that whenever you delete a record from Table A, any records in Table B that are linked to the deleted record will also be deleted. This is called *cascading delete.* Finally, the referential integrity rules could specify that whenever you modify the value of a linked [field](http://www.webopedia.com/TERM/F/field.html) in Table A, all records in Table B that are linked to it will also be modified accordingly. This is called *cascading update.*

--

2. Difference between repeating data and redundant data. Please provide a real life example.

Repeating data is when you don’t need it anymore.

Redundant is when it is printed out twice. Like FName LName LName.

3. Please don’t use X, Y, Z or A, B, C or such abstract variables. Only real world data examples are permissible.

1. Explain subtype and provide an example.

A subgrouping of the entities in an entity type that is meaningful to the organization and that shares common attributes or relationships distinct from other subgroupings. See also Generalization, Specialization, Supertype.

---

Are subgroups of the super type entity, and have unique attributes, but they will be different from each subtype

---

a subset of features in a feature class or objects in a table that share the same attributes. For example, the streets in a streets feature class could be categorized into three subtypes: local streets, collector streets, and arterial streets. Creating subtypes can be more efficient than creating many feature classes or tables in a geodatabase. For example, a geodatabase with a dozen feature classes that have subtypes will perform better than a geodatabase with a hundred feature classes. Subtypes also make editing data faster and more accurate because default attribute values and domains can be set up. For example, a local street subtype could be created and defined so that whenever this type of street is added to the feature class, its speed limit attribute is automatically set to 35 miles per hour.

---

|  |  |
| --- | --- |
| **Specialization** | The process of defining one or more subtypes (subclasses) of the supertype (superclass), and forming supertype/subtype (superclass/subclass) relationships.*See also* Attribute inheritance, Generalization. |

--

|  |  |
| --- | --- |
| **Generalization** | The process of defining a more general entity type (object class) from a set of more specialized entity types (object classes). *See also* Attribute inheritance, Specialization, Subtype, Supertype. |

===

|  |  |
| --- | --- |
| **Supertype** | A generic entity type that has a relationship with one or more subtypes. *See also* Generalization, Specialization, Subtype. |

--

|  |  |
| --- | --- |
| **Supertype/subtype hierarchy** | A hierarchical arrangement of supertypes and subtypes, where each subtype has only one supertype. |

2. Explain these terms with an example for each:

a. Partial Dependency

* + Exists when there is a functional dependence in which the determinant is only part of the primary key

-----

### Partial Dependency

Partial Dependency is a form of Functional dependency that holds on a set of attributes. It is about the complete dependency of a right hand side attribute on one of the left hand side attributes. In a functional dependency XY → Z, if Z (RHS attribute) can be uniquely identified by one of the LHS attributes, then the functional dependency is partial dependency.

**Example:**

Let us assume a relation R with attributes A, B, C, and D. Also, assume that the set of functional dependencies F that hold on R as follows;

F = {A → B, D → C}.

From set of attributes F, we can derive the primary key. For R, the key can be (A,D), a composite primary key. That means, AD →BC, AD can uniquely identify B and C. But, for this case A and D is not required to identify B or C uniquely. To identify B, attribute A is enough. Likewise, to identify C, attribute D is enough. The functional dependencies AD → B or AD → C are called as Partial functional dependencies.

**-------------------**

**In database management systems, partial dependency is a functional dependency that refers to the phenomenon where a primary key determines the outcome of another attribute or set of attributes.** It occurs when a non-key attribute of a table in a database is dependent on the value of only a part of the table’s primary key but not the entire primary key.

### FULL ANSWER

Functional dependency is also referred to as normalization, and it was introduced in 1970 by Professor Codd. It defines three normal forms (first, second and third) and is used to avoid three anomaly types that may affect databases. These three anomalies are insertion, deletion and update. An anomaly occurs when a database is incomplete, inconsistent and/or contradictory. The basic aim of normalization is to create a proper description of the attributes of particular data. It highlights relationships in different data in a table. For example, a table of students may have student name, course, admission number and grades as the fields of entry. If one or more of these fields are related, normalization ensures that any change to the details of one student does not affect the details of another. With anomalies, user cannot insert new data even when it is absolutely necessary to do so. In addition, deleting the data of one student may result in the loss of data of another. Similarly, when a record is updated, other instances of the same item are not updated.

--

Partial dependency means that a nonprime attribute is functionally dependent on part of a candidate key. (A nonprime attribute is an attribute that's not part of any candidate key.)

For example, let's start with R{ABCD}, and the functional dependencies AB->CD and A->C.

The only candidate key for R is AB. C and D are a nonprime attribute. C is functionally dependent on A. A is part of a candidate key. That's a partial dependency.

---

|  |  |  |
| --- | --- | --- |
| |  | | --- | |  | | Every FD that holds is either full or partial; keys don't enter into it. – [philipxy](http://stackoverflow.com/users/3404097/philipxy" \o "6,106 reputation) [Sep 13 '14 at 20:13](http://stackoverflow.com/questions/25747802/partial-dependencydatabases#comment40404877_25751643) |
| |  |  | | --- | --- | | 1 |  | | C is partially dependent on AB, ie AB->C is a partial FD, because C is dependent on a proper subset of AB, A. C is fully dependent on A, ie A->C is a full FD, because C is not functionally dependent on a proper subset of A. – [philipxy](http://stackoverflow.com/users/3404097/philipxy" \o "6,106 reputation) [Sep 14 '14 at 4:32](http://stackoverflow.com/questions/25747802/partial-dependencydatabases#comment40410100_25751643) |

---

|  |  |
| --- | --- |
|  | **Partial dependency** implies is a situation where a non-prime attribute(An attribute that does not form part of the *determinant*(Primary key/Candidate key)) is ***functionally dependent*** to a portion/part of a primary key/Candidate key. |

b. Repeating Data

Repeating data is when the data is used repeated for different projects for example Last name can be repeated out over and over but is relate to different data. Still valid data.

* + Group of multiple entries of same type can exist for any single key attribute occurrence

Redundant is when the same data is stored twice for different depts for example. Marketing and Engineer dept. stores the same info

c. Transitive Dependency

* + Exists when there are functional dependencies such that X → Y, Y → Z, and X is the primary key

In Database Management System, a **transitive dependency** is a functional **dependency** which holds by virtue of **transitivity**. A **transitive dependency** can occur only in a relation that has three or more attributes. Let A, B, and C designate three distinct attributes (or distinct collections of attributes) in the relation.

transitive dependency

An indirect relationship between data elements in a database. The rule is essentially that A is a transitive dependency of C (A->C) if A is functionally dependent on B (A->B), and B is functionally dependent on C (B->C) but not on A (B not->A).  
  
**For Example**  
Social security number is a transitive dependency of date-of-birth (SSN->DOB), because social security number is dependent on employee name (SSN->EMP), and employee name is dependent on date-of-birth (EMP->DOB) but not on social security number (there may be more than one John Smith or Maria Gomez). See [functional dependency](http://www.pcmag.com/encyclopedia/term/65917/functional-dependency).

**Definition:**Transitive dependencies occur when there is an indirect relationship that causes a[functional dependency](http://databases.about.com/library/glossary/bldef-functdep.htm).  
  
For more detail, read [What is a Database Dependency?](http://databases.about.com/od/specificproducts/a/Database-Dependency.htm)

**Examples:**For example, ”A -> C” is a transitive dependency when it is true only because both “A -> B” and “B -> C” are true.

3. On which side would a foreign key go in a parent child (1:M) relationship?

Child side

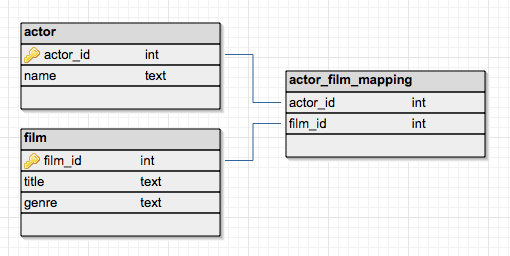
4. What is an associative entity? Please provide an example of how M:N relationship can be handled in the design process through a bridge/associative entity? Provide an illustration different from the movie and star relationship discussed in the class. Make an ERD for your example.

An **associative entity** is a term used in [relational](https://en.wikipedia.org/wiki/Relational_model) and [entity–relationship](https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model) theory. A relational database requires the implementation of a base relation (or base table) to resolve[many-to-many relationships](https://en.wikipedia.org/wiki/Many-to-many_(data_model)). This kind of base relation is called an **associative table**.

[https://upload.wikimedia.org/wikipedia/en/1/14/Associate_Entity.png](https://en.wikipedia.org/wiki/File:Associate_Entity.png)

An associative entity (using [Chen](https://en.wikipedia.org/wiki/Peter_Chen) notation)

As mentioned above, associative entities are implemented in a database structure using **associative tables,** which are tables that can contain references to columns from the same or different database tables within the same database.

[](https://en.wikipedia.org/wiki/File:Mapping_table_concept.png)

An associative (or junction) table maps two or more tables together by referencing the primary keys of each data table. In effect, it contains a number of foreign keys, each in a many-to-one relationship from the junction table to the individual data tables. The PK of the associative table is typically composed of the FK columns themselves.

Associative tables are colloquially known under many names, including **association table**, **bridge table**, **cross-reference table**, **crosswalk**, **intermediary table**, **intersection table**, **join table**, **junction table**, **link table**, **linking table**, **many-to-many resolver**, **map table**, **mapping table**, **pairing table**, or **transition table**.

What is the difference between an associative entity and an associative relationship attribute? In my book titled, Modern Database Management (Hoffer, 11th edition), the author states that there is a difference between the two. Though, it doesn't really explain why there's a difference, instead, it just gives examples of how they're different.

From what I gather, a relationship that has a **single** attribute associated with it is an associative relationship attribute and is denoted with a dashed line to a rounded-corner rectangle with that attribute inside that rectangle. Whereas an associative entity is **more than one** attribute that describe the relationship. Both can only be used for many-to-many relationships in ER diagramming. Is my thought process correct?

Edit - Also, I forgot to mention, with associative entities, they also have an attribute that is a unique identifier, at least, this is what I think.

An [associative entity](http://en.wikipedia.org/wiki/Associative_entity) is the table that associates two other tables in a many to many relationships.

An associative relationship attribute is an attribute of the associative entity that exists because of the many to many relationships.

Here's an example. Let's suppose we have the following tables.

User

----

User ID

User Login Name

User Name

User Password

Permission

----------

Permission ID

Permission Name

Permission Description

Ok, we have a many to many relationships between User and Permission. A user can have more than one permission, and a permission can be shared between many users.

So, we create an associative entity.

User Permission

--------------

User ID

Permission ID

Permission Granted Time Stamp

The permission granted time stamp is an associative relationship attribute. It would not fit in the User table nor the Permission table. It's an attribute of the association.

5. What is a weak entity? Provide an example of how repeating data can be handled by creating a weak entity. We have learned how to handle the case of major, a repeating attribute in Student table, therefore, please provide a different example. Make an ERD for your example.

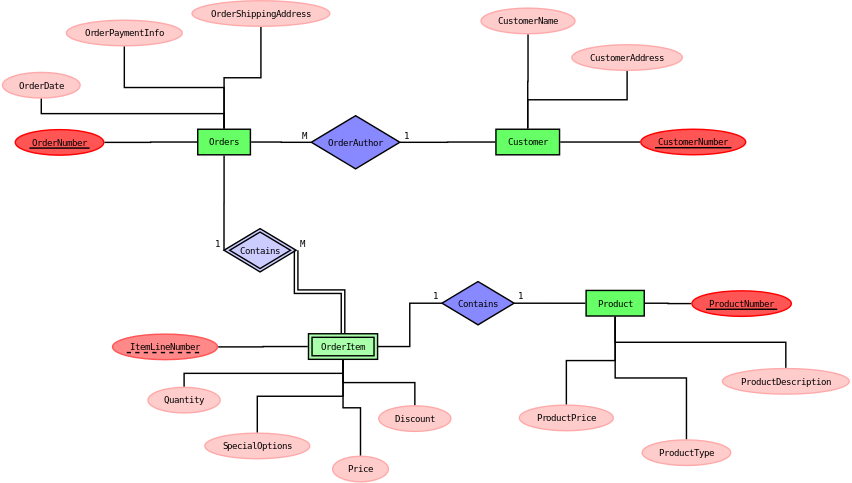
a **weak entity** is an entity that cannot be uniquely identified by its attributes alone; therefore, it must use a [foreign key](https://en.wikipedia.org/wiki/Foreign_key) in conjunction with its attributes to create a [primary key](https://en.wikipedia.org/wiki/Primary_key). The foreign key is typically a primary key of an entity it is related to.

In [entity relationship diagrams, ER diagrams](https://en.wikipedia.org/wiki/Entity_relationship_diagram) a weak entity set is indicated by a bold (or double-lined) rectangle (the entity) connected by a bold (or double-lined) type arrow to a bold (or double-lined) diamond (the relationship). This type of relationship is called an *identifying relationship* and in [IDEF1X](https://en.wikipedia.org/wiki/IDEF1X) notation it is represented by an oval entity rather than a square entity for base tables. An identifying relationship is one where the primary key is populated to the child weak entity as a primary key in that entity.

In general, (though not necessarily) a weak entity does not have any items in its primary key other than its inherited primary key and a sequence number. There are two types of weak entities: [associative entities](https://en.wikipedia.org/wiki/Associative_Entities) and [subtype entities](https://en.wikipedia.org/w/index.php?title=Subtype_entities&action=edit&redlink=1). The latter represents a crucial type of [normalization](https://en.wikipedia.org/wiki/Database_normalization), where the [super-type entity](https://en.wikipedia.org/w/index.php?title=Super-type_entity&action=edit&redlink=1) inherits its attributes to entities based on the value of the [discriminator](https://en.wikipedia.org/wiki/Discriminator).

EXAMPLE:

Consider a database that records customer orders, where an order is for one or more of the items that the enterprise sells. The database would contain a table identifying customers by a customer number ([primary key](https://en.wikipedia.org/wiki/Primary_key)); another identifying the products that can be sold by a product number ([primary key](https://en.wikipedia.org/wiki/Primary_key)); and it would contain a pair of tables describing orders.



<https://en.wikipedia.org>

ecomputernotes.com

[www.datanamic.com](http://www.datanamic.com)

<https://opentextbc.ca>

stackoverflow.com

databases.about.com

[www.pcmag.com](http://www.pcmag.com)

<https://exploredatabase.blogspot.com/>

### [**Reference.com**](https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&cad=rja&uact=8&ved=0ahUKEwi5yb7W4sHPAhUl2IMKHSeEB48QFggnMAI&url=https%3A%2F%2Fwww.reference.com%2Ftechnology%2Fpartial-dependency-91beb2fa8c8354c5&usg=AFQjCNHZH9O5LLSvAIblDp_S7uH-7X8WDw&sig2=uNodGPAiSF4zWJ8WmZyHvw)

[www.sciencedirect.com](http://www.sciencedirect.com)

[www.prenhall.com](http://www.prenhall.com)

<https://learndatamodeling.com>

<https://msdn.microsoft.com>

sqlmag.com

support.esri.com

geverest.umn.edu/

dba.stackexchange.com

[www.webopedia.com](http://www.webopedia.com)