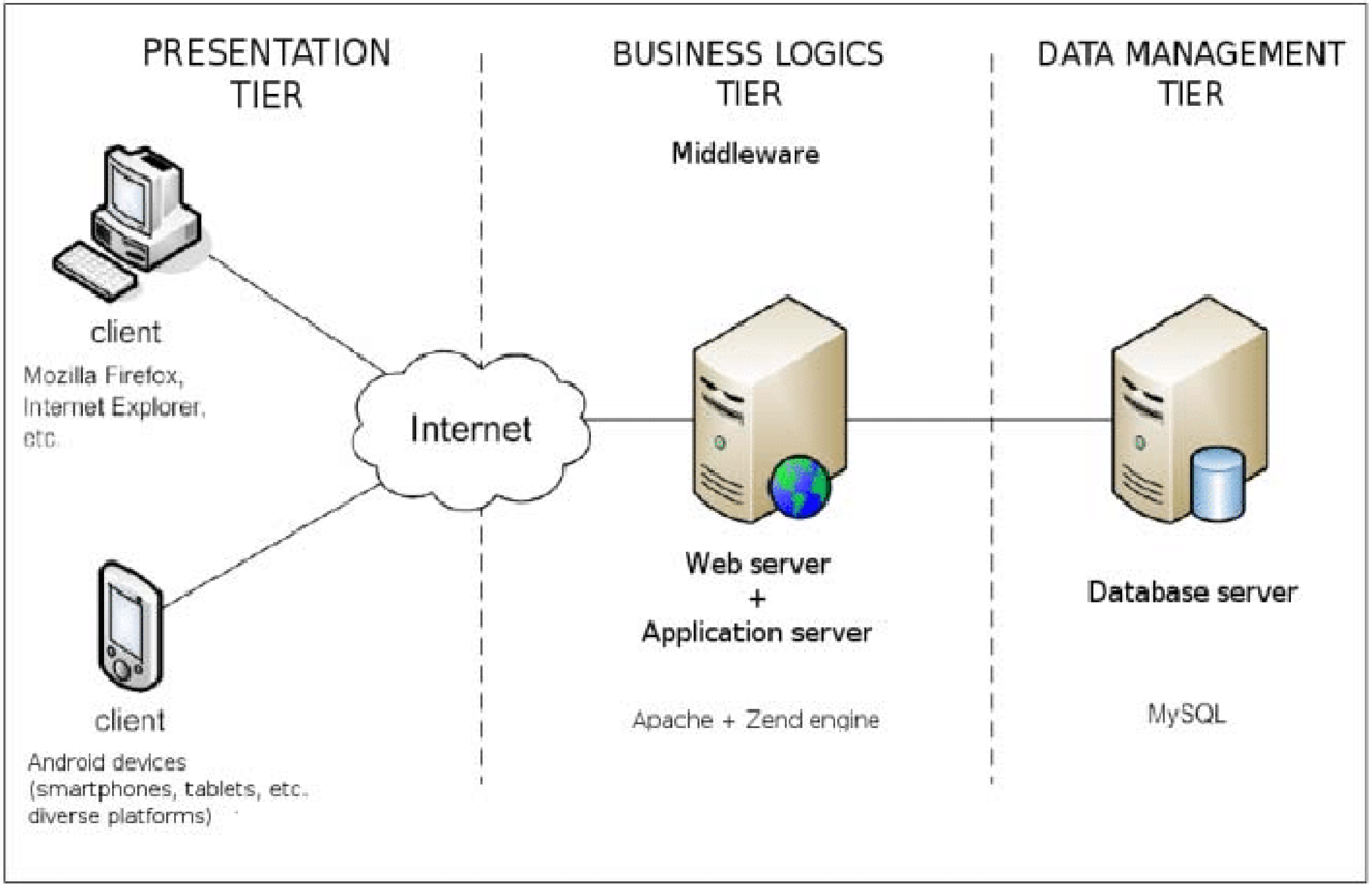
Edith Cowan University
School of Science

CSI6207
Systems Analysis and Databese Design

Normalisation and Data Modeling

3 Tier Architecture



Entities, Databases and Tables

- The "Things" used in object modeling often contain state information that needs to be stored.
- One of the most common implementations of this is to use a **Relational Database** to store data.
- Within a database, each of these "things" is called a **data entity**
- Within a database, each of these "**data entities**" is stored in a **table** and individual instances of these entities are stored as a **row** of that table

Student#	Student Name	Address	DoB
0972343	Eric Cartman	1 Normalisation Rd	12/10/1989
0982342	Kyle Broflowski	12 Smith Street	30/03/1986
2013442	Stan Marsh	2A Evergreen Terrace	30/03/1986
0972345	Tab Cartman	1 Normalisation Rd	17/10/1992

An example of a "Student" entity and some of its instances:

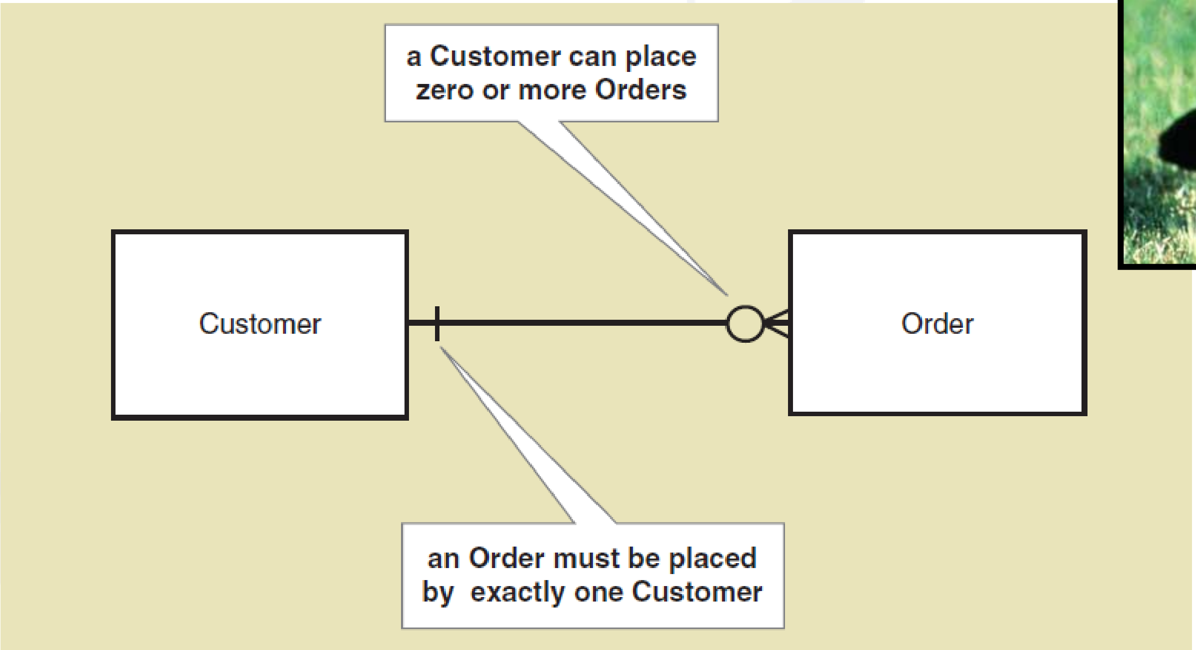
ER Diagram

Entity-Relationship Diagrams ERD

- ERDs have been used for many years to develop entities and their relationships that can be used in database development
- ERD models are not UML models and do not use standard UML notation, although they are similar to class diagrams.
- ERD models are not as expressive as UML models for modeling objects but are best suited to modeling databases
  - They do not model generalization/specialization well
  - They do not model whole/part well

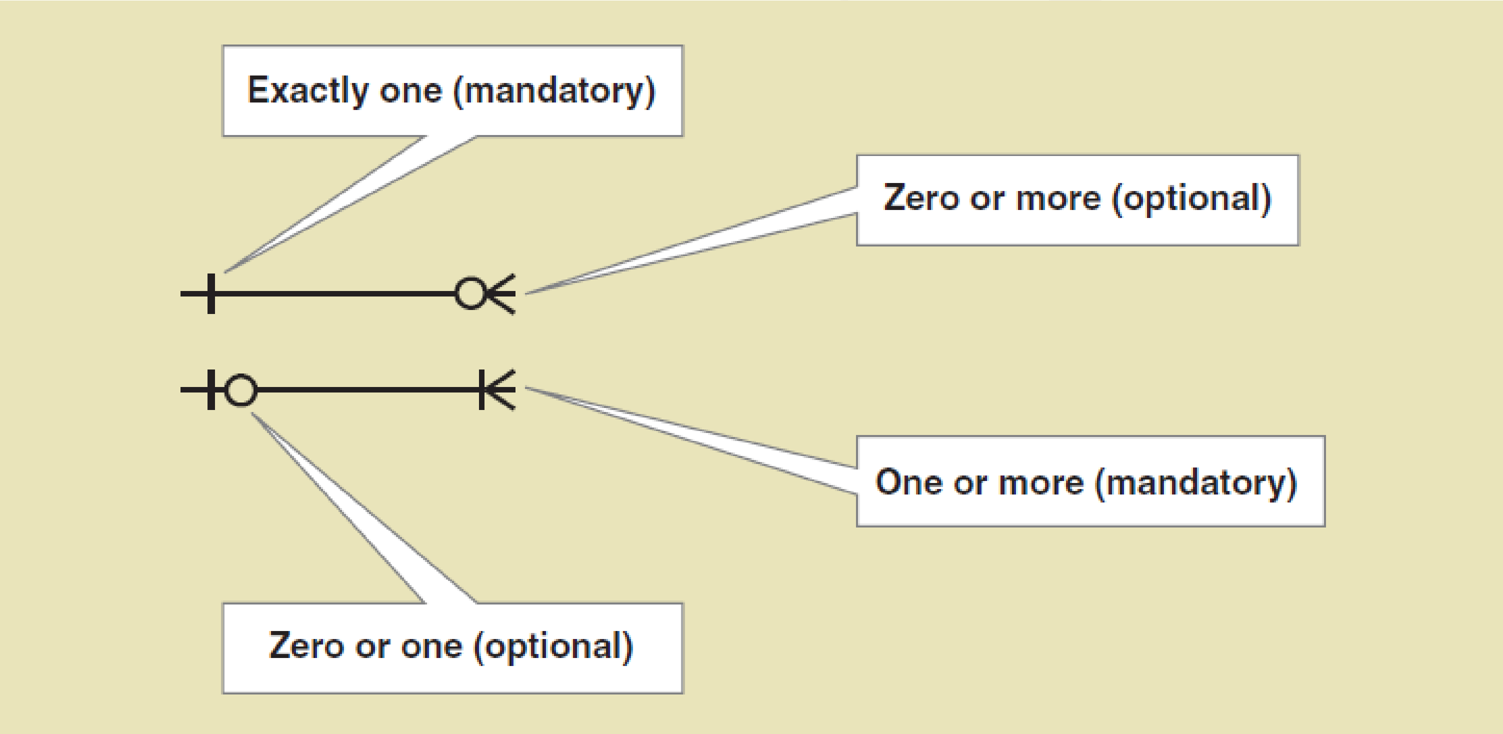
Example of ERD Notation

- ERD Models normally use “crows feet” notation to show cardinality



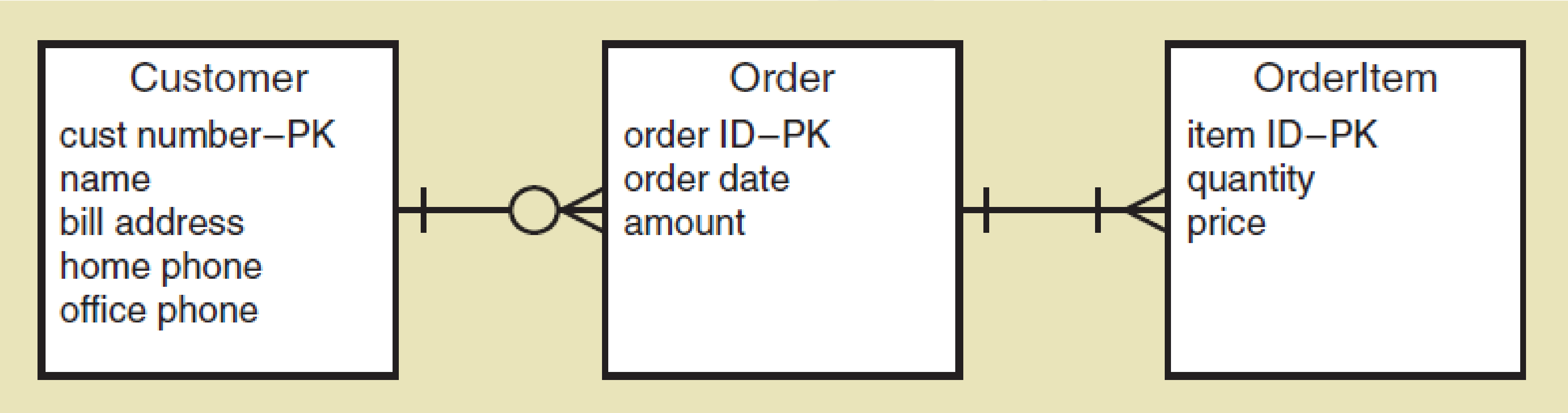
ERD Cardinality Symbols

- Examples of crows feet notation for various cardinalities



Expanded ERD with Attributes

- ERD with cardinalities and attributes
- There are several different notation methods for attributes in ERD models
- This notation places attributes within data entities



Keys

- The term “key” refers to one or more attributes (columns) in an entity (table) that is used to:
  - Uniquely identify each row in the table so that it can be referred to - known as a ***Primary Key***

FIGURE 16-1: A TELEPHONE BOOK TABLE

Last name	First name	Address	Phone
Abbott	William	123 Oak Lane	490-8920
Ackerman	Kimberly	467 Elm Drive	787-2781
Adams	Stanley	8120 Pine Street	787-0129
Adams	Violet	347 Oak Lane	490-8912
Adams	William	12 Second Street	490-3667

  - Link a pair of columns between tables - known as a ***Foreign Key***
  - The identification of the correct keys is central to normalisation and databases

Primary Keys (PK)

- A primary key is… *one or more columns in a table that uniquely identifies each row*

- Usually denoted by underlining the name of the primary key attribute, e.g.

Student#	Student Name	Address	DoB
0972343	Eric Cartman	1 Normalisation Rd	12/10/1989
0982342	Kyle Broflowski	12 Smith Street	30/03/1986
2013442	Stan Marsh	2A Evergreen Terrace	30/03/1986
0972345	Tab Cartman	1 Normalisation Rd	17/10/1992

- The primary key attribute(s) must be unique – two rows cannot have the same primary key value

Foreign Keys (FK)

- A foreign key is… *a column in a table that refers to the primary key of another table*
  - Unlike primary keys, foreign keys do not need to be unique - since a row in one table may relate to multiple rows in another table
  - e.g. one course has many students enrolled in it (hence, the student table has a “course_code” FK column – which is the PK of the course table)

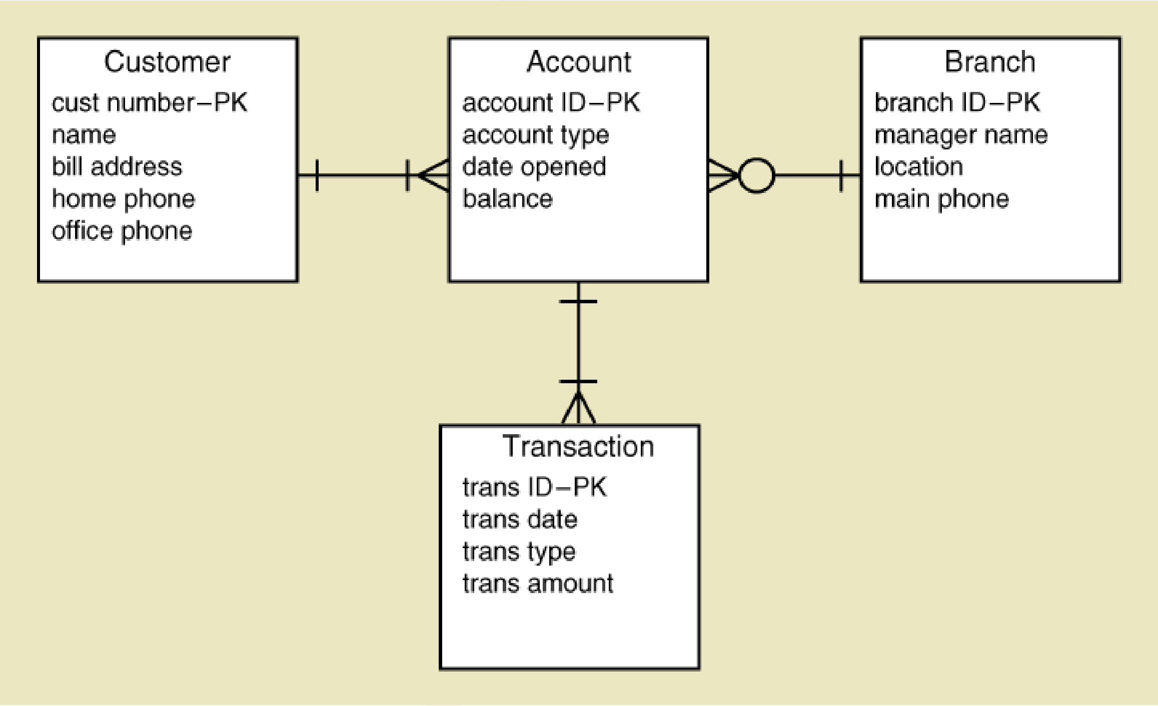
- Foreign keys are usually denoted by *italicising* the name of their attributes

- This allows us to define relationships between tables

- A foreign key should have the same domain (data type, range constraints, etc) of the primary key it references

An ERD for a Bank

- Quick Quiz
  - What are the key fields?
  - How many accounts can a customer have?
  - How many branches can a customer be assigned to?
  - How many customers can a branch have?



Primary and Foreign Key Relationships

Student Table

Student#	Student Name	Address	DOB
0972343	Eric Cartman	1 Normalisation Rd	12/10/1989
0982342	Kyle Broflowski	12 Smith Street	30/03/1986
2013442	Stan Marsh	2A Evergreen Terrace	18/09/1990
0972345	Tab Cartman	1 Normalisation Rd	17/10/1992

Student = (<u>Student#</u>, Student Name, Address, DOB)

Unit Table

Unit Code	Unit Name
CSG1207	Systems & Database Design
CSG1209	Application development
WIN2441	Wine Appreciation
TAV1234	Tavern Studies

Unit = (<u>Unit Code</u>, Unit Name)

Enrolment Table

Enrolment#	*Student#*	*Unit Code*	Semester	Year	Mark	Grade
1	0972343	CSG1207	1	2018	37	N
2	0982342	CSG1207	1	2018	84	HD
3	0982342	TAV1234	1	2018	62	CR
4	2013442	WIN2441	2	2018	*NULL*	*NULL*
5	0972343	CSG1207	2	2018	*NULL*	*NULL*

Enrolment = (<u>Enrolment#</u>, Student#, Unit Code, Semester, Year)

- [illegible]
- [illegible]

Compound (Primary) Keys

- If two or more columns must be combined to uniquely identify each row, it is a **compound key**
- Denoted in the same way as other primary keys…

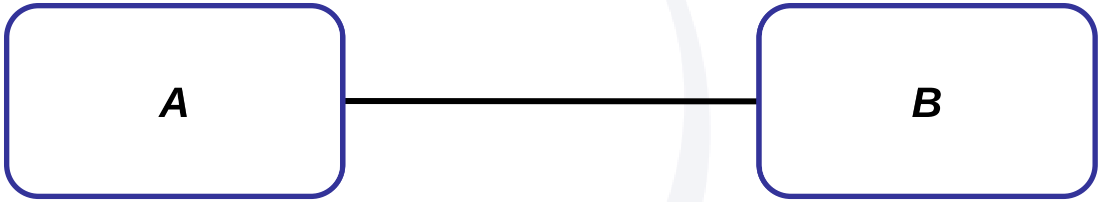
(<u>Student#</u>, <u>Unit Code</u>, Mark, Grade)

<u>Student#</u>	<u>Unit Code</u>	Mark	Grade
0972343	CSG1207	41	N
0982342	CSG1207	63	CR
2013442	CSI2441	96	HD
0972343	CSI2441	74	D

- Either of the two fields may have duplicate entries, but *when combined they must be unique*

One-to-One Relationship

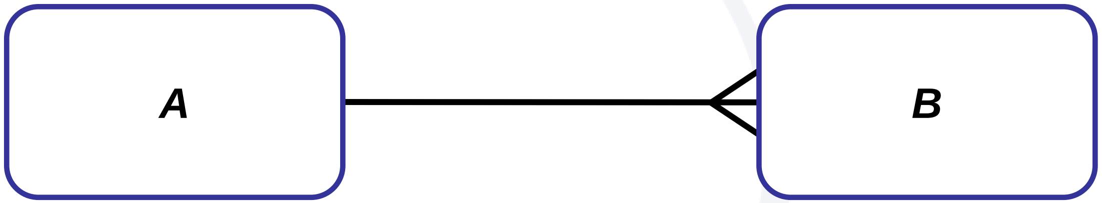
- Signifies that **for each instance of entity A there is one related instance of entity B**



- Shown via a simple straight line joining two entities
  - Where this type of relationship exists the entities joined may have the same primary keys and thus it is often questioned whether or not the two entities should be separate
  - Thus while the model acknowledges the 1:1 existence it is not often used or seen in full-scale relational databases

One-to-Many Relationship

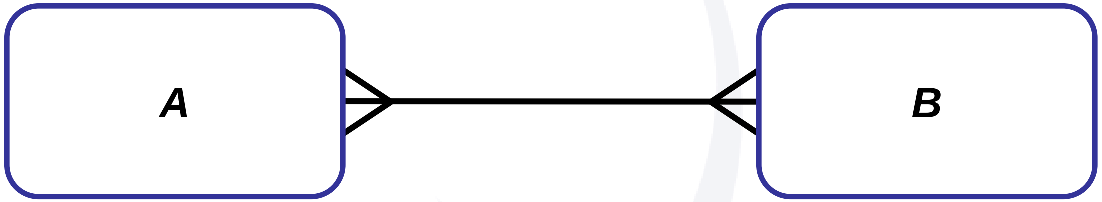
- Signifies that **for each instance of entity A there is one or more related instances of entity B**



- Shown via a straight line with a *crow's foot* on the *many* side of the relationship
  - Where this type of relationship exists, the *primary key* of the *1-side* entity of the relation is present as a *foreign key* in the *many-side* entity.
  - This is the *most common relationship* that exists between two entities; Almost all relationships in most scenarios will be 1:M

Many-to-Many Relationship

- Signifies that **for each instance of A there is one or more related instances of entity B *and vice versa***



- Shown via a line with *crow's feet* on *both ends*.
  - Where this type of relationship exists, the primary keys of each entity exists as a foreign key in the other (but how?)

  - Due to the complexity in actually implementing the logic behind this type of relationship they must be *resolved* in an ER diagram
    - Cannot *implement* a M:M relationship in an actual database

Resolving a M:M Relationship

- ***A many to many relationship exists between the entities of student and unit – many students in one unit, one unit has many students. This cannot be managed with PK/FKs…***
- ***The relationship must be resolved, by introducing a new entity between them, hence forming two 1:M relationships***

StudentNum	Student Name
0972343	Eric Cartman
0982342	Kyle Broflowski
2013442	Stan Marsh
3992342	Kenny McCormack

Unit Code	Unit Name
CSG1207	Systems & Database Design
CSI2441	Application Development
CSG2431	Interactive Web Development

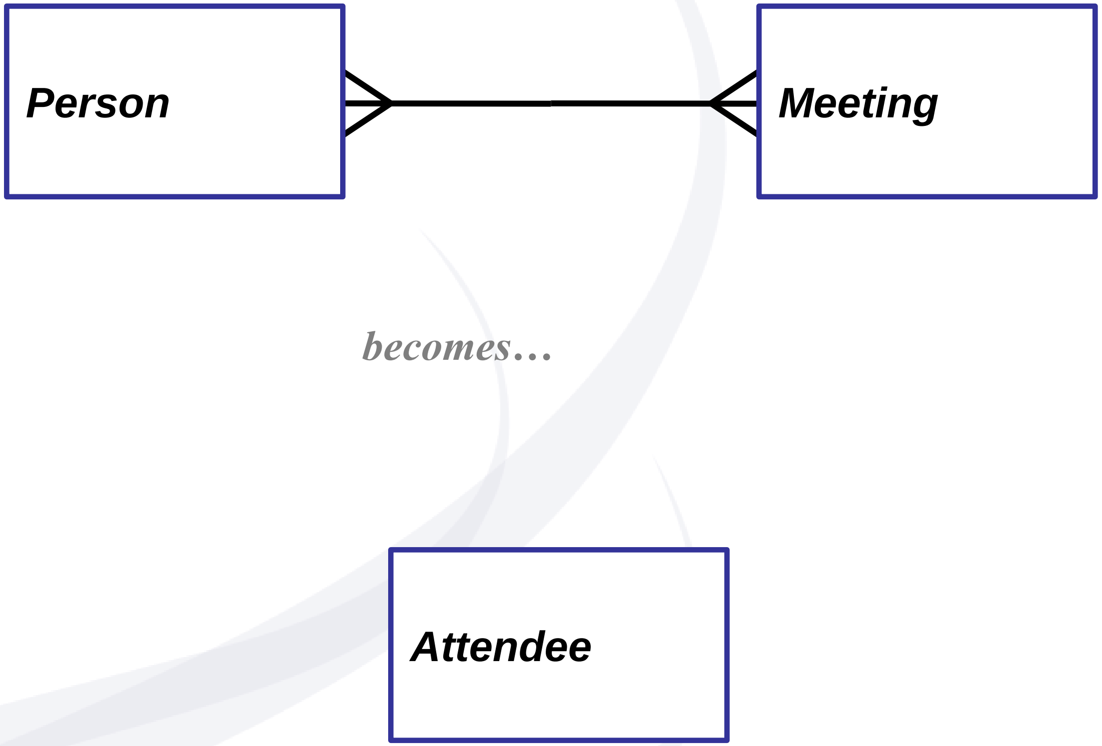
StudentNum	Student Name
0972343	Eric Cartman
0982342	Kyle Broflowski
2013442	Stan Marsh
3992342	Kenny McCormack

StudentNum	*Unit Code*
0972343	CSG1207
0972343	CSI2441
0982342	CSG1207
0982342	CSI2441
2013442	CSI2441
2013442	CSG2431
…	…

Unit Code	Unit Name
CSG1207	Systems & Database Design
CSI2441	Application Development
CSG2431	Interactive Web Development

Resolving a M:M Relationship

- **Another Example**



Resolving a M:M Relationship

- Three steps:
  1. A new entity is created as an intermediary between the two existing entities
  2. The original entities both form a 1:M relationship with the intermediary entity
  3. The new entity inherits the primary key attributes of the two original entities (as foreign keys)
     - These may also become a compound primary key for the new entity, but it is common to create an auto-incrementing integer field to act as a primary key instead
     - These may be the *only* attributes in the new entity, but not always – e.g. Items in an order will have a quantity attribute

Naming the Intermediary Entity

- When naming the newly created intermediary entity, firstly consider if the new entity is akin to any ‘real life’ object or term. e.g. Attendee, Enrolment, Appointment, etc
  - Specific:

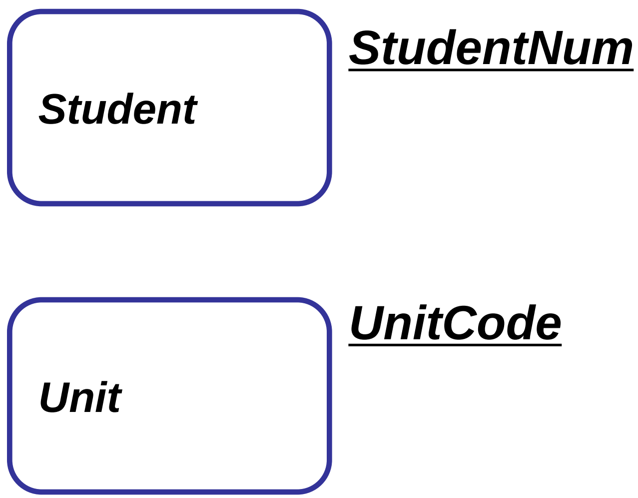
- If no specific term is apparent, a common technique is to name the new entity with a hybrid of the two original entities. e.g. OrderItem, LawyerCase, etc
  - Hybrid:

Attributes

- To add detail and value to an ER diagram, **attributes** of entities should be shown
  - i.e. what data is stored about each entity

- A good place to start is with the **primary keys** – by this stage you should have a quite clear idea of likely primary keys and their correct placement

- Primary keys are usually depicted alongside their associated entity and underlined



Attributes

- **Non-key attributes** are then listed in order of priority/logic after the primary key attribute(s), separated by commas

Student

StudentNum, Surname, FirstName, Gender, DoB

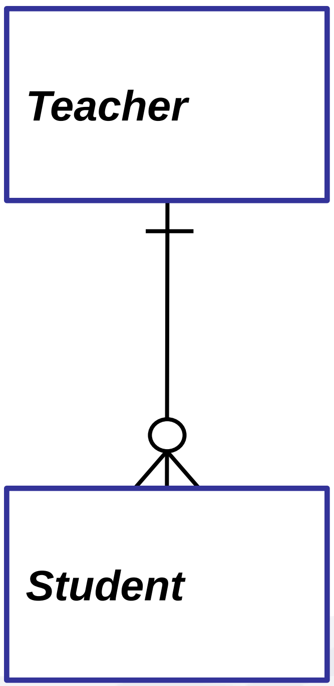
UnitCode, UnitTitle, CreditPoints, Description

Unit

Attributes

- Foreign key attributes are shown *italicised*
- Add the primary key of the “one” side to the “many” side

***<u>TeacherNum</u>*, Surname, FirstName**

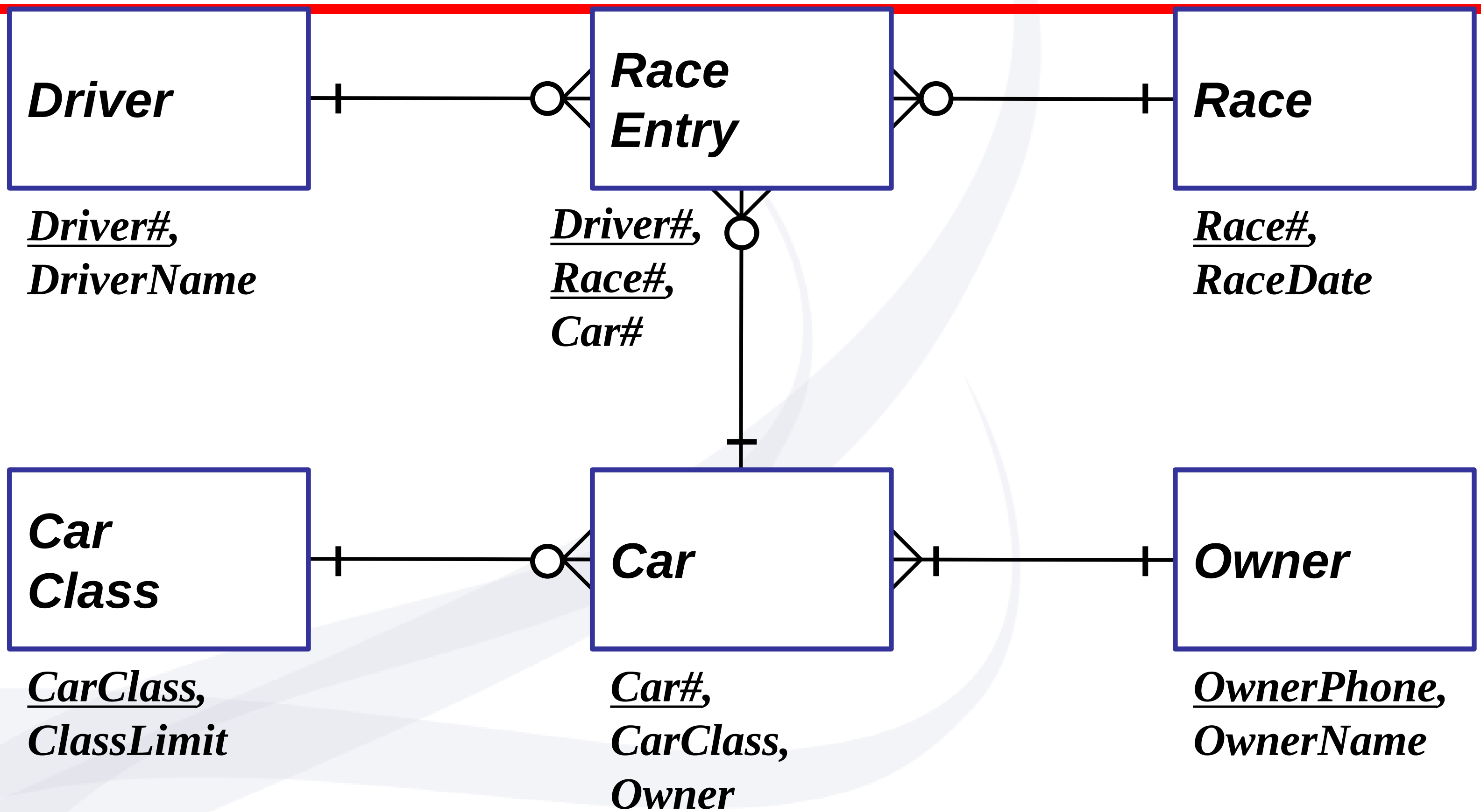


***<u>StudentNum</u>*, Surname, FirstName, Gender**

***<u>StudentNum</u>*, *TeacherNum*, Surname, FirstName, Gender**

- Remember: Foreign keys always flow from **1 → Many**, never from Many → 1

A complete ER Model



Data Normalisation

Anomalies when data is not Normalised

- ***Three common types of anomalies:***

An **Update Anomaly** exists when one or more instances of duplicated data is updated, but not all. For example, consider Jones moving address - you need to update all instances of Jones's address.

StudentNum	CourseNum	Student Name	Address	Course
S21	9201	Jones	Edinburgh	Accounts
S21	9267	Jones	Edinburgh	Accounts
S24	9267	Smith	Glasgow	physics
S30	9201	Richards	Manchester	Computing
S30	9322	Richards	Manchester	Maths

A **Delete Anomaly** exists when certain attributes are lost because of the deletion of other attributes. For example, consider what happens if Student S30 is the last student to leave the course - All information about the course is lost.

StudentNum	CourseNum	Student Name	Address	Course
S21	9201	Jones	Edinburgh	Accounts
S21	9267	Jones	Edinburgh	Accounts
S24	9267	Smith	Glasgow	physics
S30	9201	Richards	Manchester	Computing
S30	9322	Richards	Manchester	Maths

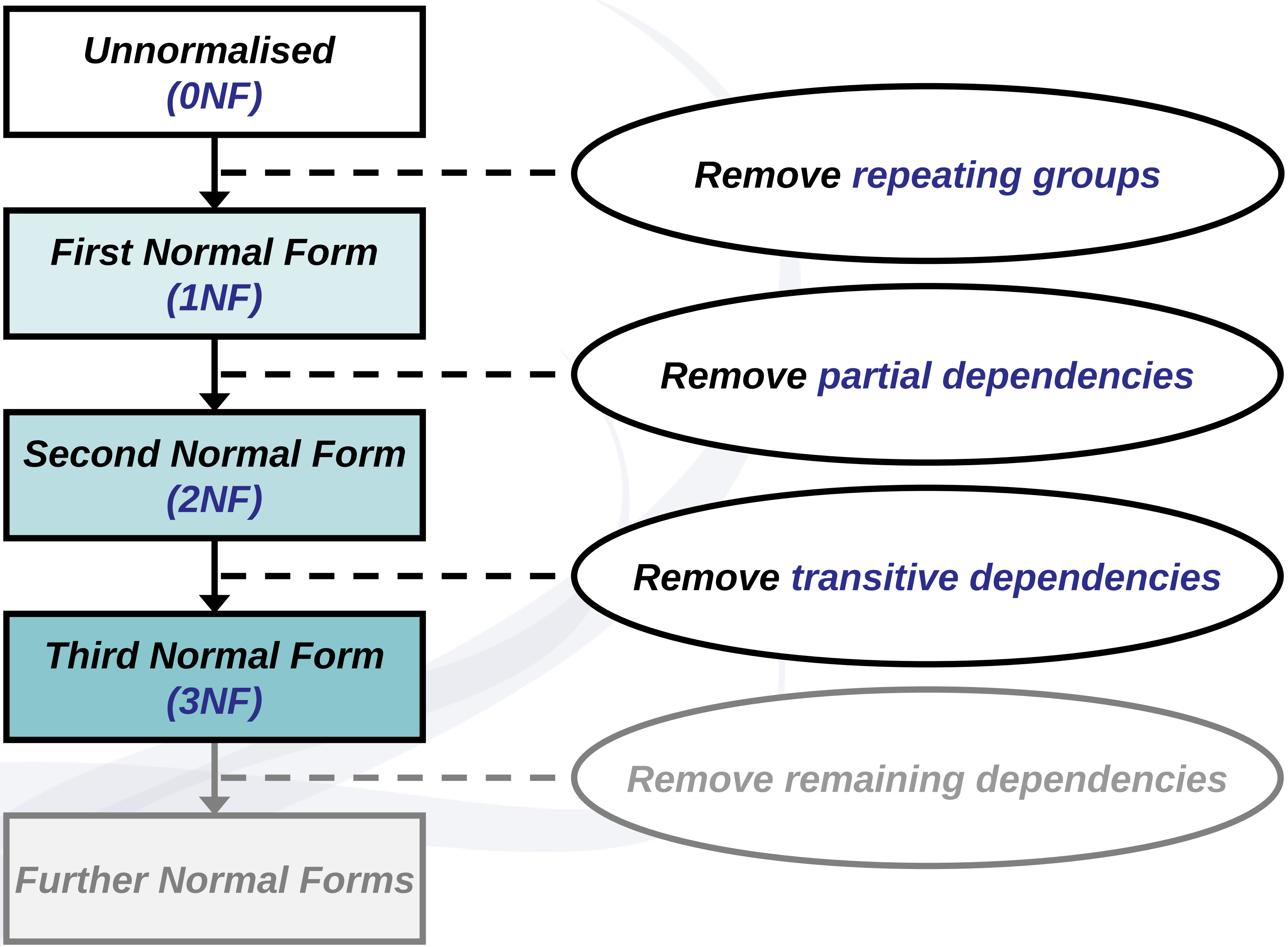
An **Insert Anomaly** occurs when certain attributes cannot be inserted into the database without the presence of other attributes. For example this is the converse of delete anomaly - we can't add a new course unless we have at least one student enrolled on the course.

Normalisation Process

- The normalisation process follows a standard series of steps, known as Normal Forms (NF):
  - Gather the unnormalised data set (0NF)
  - Convert to first normal form (1NF)
  - Convert to second normal form (2NF)
  - Convert to third normal form (3NF)
  - (4NF & 5NF exist, but typically 3NF is “normalised”)

- ***NOTE:*** *Unless each step is carried out properly, the next step will be flawed, i.e. unless a data set is in first normal form it will not be in a valid second or third normal form*

Stages of Normalisation



When to End the Process?

- 2NF is better than 1NF, 3NF is better than 2NF
- For most business database design purposes, **3NF** is as far as we need to normalise
- 3NF is typically considered “normalised”, and databases in 3NF usually already in 4NF & 5NF
- Highest level of normalisation is not always most desirable

Relational Symbolic Notation

- During the normalisation process, we use relational symbol notation, (e.g. **R1**, **R11, R12**) to represent relations, rather than relation names
- When a relation, say **R1**, is split into two or more relations, the new relations will be named **R11**, **R12**, **R13**…
  - Like labelling sections of a report, you add another number each time you split into a further/deeper "level" of relations
  - e.g., if **R1223** is split, the resultant relations will be named **R12231**, **R12232**, **R12233**…
  - This makes it easy for us to trace the relation splitting if we find something wrong in some late stage of the normalisation process
- When showing progress through the normal forms, use ~~strikethrough~~ to indicate when a relation has been split

Gathering Unnormalised Data Sets

- The first step of normalisation involves gathering or identifying an unnormalised data set – a collection of all the attributes (their names, not the data itself) that need to be stored

- Where groups of attributes exist in a nested manner, they are known as repeating groups – "*tables within tables*"
  - A repeating group is a set of attributes that can have more than one value for a given primary key (value)
  - i.e. For a single instance of the "outer" attributes, there can be multiple instances of a group of "inner" attributes

- This step is often rushed, but is perhaps the most critical of the entire normalisation process.
  - If the unnormalised data set contains errors then it is more than likely that these errors will be carried throughout the entire normalisation resulting in possible data anomalies

Steps from 0NF to 1NF – Example

Sales Invoice

Invoice # : 254186
Order Date: 29/6/09

Customer #: 78901
Customer Name: Fred Bloggs
Customer Phone #: 9370 6111
Customer Address: 3 Uphill Rise, Ferndale, WA 6303

Item #	Description	Qty	Unit Price	Subtotal
9898	Bearing, Ball	25	$2.50	$62.50
9999	Bearing, Roller	10	$5.00	$50.00
8888	Seal, shaft	10	$3.00	$30.00
777	Glasses, Safety	10	$10.00	$100.00
1555	Punch, 5mm	1	$4.00	$4.00

Total: $246.50

R1* = *(Invoice #, Order Date, Customer #, Name, Phone, Address,* *{Item #, Description, Qty, Unit Price})

- *Note the use of { } to show repeating groups*

Steps from 0NF to 1NF – Example

1. Firstly, splitting R1 into two relations by removing the biggest repeating group, leaving you with two relations:

"outer" relation

R11 = (Invoice #, Order Date, Customer #, Name, Phone, Address**)**

R12 = (Item #, Description, Qty, Unit Price**)**

"inner" relation

- You can switch the relation names (R11 and R12) if you like

- If there were multiple nested repeating groups in R1, then R12 would still contain repeating groups at this point
  - We'll go over the example again with the alternate R1 structure to demonstrate this later

Steps from 0NF to 1NF – Example

2. Identify the Primary Key (PK) of the "outer" relation (R11):

   R11 = (<u>Invoice #</u>, Order Date, Customer #, Name, Phone, Address**)**

   R12 = (Item #, Description, Qty, Unit Price**)**

- A primary key is an attribute (or attributes) that uniquely identifies each instance of a relation (i.e. row of a table)
  - Each invoice will have a different **Invoice #**, even if some of them have the same customer details
  - PKs are depicted via <u>underlining</u>
  - May need to combine multiple attributes (compound primary key)
  - If no appropriate primary key attribute exists in the original data, it is reasonable to add one if it is logical that it would exist
  - *Remember that throughout the normalisation process, all relations should have a valid primary key at all times*

Steps from 0NF to 1NF – Example

3. Then add a copy of the Primary Key of the “outer” relation into the “inner” relation as a Foreign Key (FK):

    R11 = (<u>Invoice #</u>, Order Date, Customer #, Name, Phone, Address**)**

    R12 = (Item #, *Invoice #*, Description, Qty, Unit Price**)**

   - This serves to *preserve the relationship* between the relations, i.e. keeping track of which items were ordered in which invoice

   - For each instance of R12, the value of the Invoice # attribute identifies the corresponding instance of R11

   - FKs are depicted in *italics*

Steps from 0NF to 1NF – Example

4. Finally, identify the PK of the "inner" relation (R12) :

   R11 = (<u>Invoice #</u>, Order Date, Customer #, Name, Phone, Address**)**

   R12 = (<u>Item #, *Invoice #*</u>, Description, Qty, Unit Price**)**

- Item # alone *cannot* uniquely identify each instance of R12, since each instance represents an *item ordered in an invoice*
  - i.e. The same Item # may appear in multiple rows of the table, since it was ordered in multiple invoices – with a different Invoice # and qty each time
  - The **combination of Item # and Invoice #** *will be unique* though, since an item can only be ordered once per invoice (the qty determines how many)
  - Each instance of this relation can now be uniquely identified – there will only ever be one instance of a *certain item* within a *certain invoice*

  - In this case we have ended up with a compound PK for R12, involving the Invoice # FK we introduced. This is *not* always the case!

Steps from 0NF to 1NF – Example

- Final relations at 1NF:

  R11 = (<u>Invoice #</u>, Order Date, Customer #, Name, Phone, Address**)**

  R12 = (<u>Item #, *Invoice #*</u>, Description, Qty, Unit Price**)**

- This process must be repeated for any remaining repeating groups in the new relations
  - There are no more in this example

- Remember to ensure that all relations have a valid primary key, and a foreign key to preserve relationships

- Note that Invoice # in R12 is both a *FK attribute*, as well as being *part of the compound PK* of the relation

Example Summary

~~**R1 = (<u>Invoice #</u>**, *Order Date, Customer #, Name, Phone, Address,* **{<u>Item #</u>,** *Description, Qty, Unit Price***})**~~

R12 = (*Item #, Description, Qty, Unit Price****)***

R11 = (<u>Invoice #</u>*, Order Date, Customer #, Name, Phone, Address****)***

R12 = (*Item #***, ***Invoice #, Description, Qty, Unit Price****)***

R12 = (<u>Item #</u>, <u>Invoice #</u>*, Description, Qty, Unit Price****)***

1NF:

R11 = (<u>Invoice #</u>*, Order Date, Customer #, Name, Phone, Address)*

R12 = (<u>Item #</u>, <u>Invoice #</u>*, Description, Qty, Unit Price****)***

Second Normal Form (2NF)

- **2NF:** A relation is in 2NF if it is in 1NF, and every non-key attribute is *fully dependent on the entire primary key*

- To go from 1NF to 2NF, you must *resolve all partial dependencies* that exist in the relations

  – A partial dependency occurs when an attribute is not wholly-dependent on the *entire* primary key

TABLE_PURCHASE_DETAIL

Customer ID	Store ID	Purchase Location
1	1	Los Angeles
1	3	San Francisco
2	1	Los Angeles
3	2	New York
4	3	San Francisco

  – Can only occur in relations that have more than one primary key attribute (i.e. a compound key)

  – Often occurs when the primary key was expanded to include the foreign key after a relation with a repeating group was split

Steps from 1NF to 2NF – Example

- Upon reaching 1NF, the relations in our example are:

  R11 = (<u>Invoice #</u>, Order Date, Customer #, Name, Phone, Address**)**

  R12 = (<u>Item #</u>, <u>*Invoice #*</u>, Description, Qty, Unit Price**)**

  – R11 does not have a compound key, so it has no partial dependencies

  – R12 has a compound key, and we can see that the Description and Unit Price attributes depend only on Item # (Qty depends on the whole key)

1. First, remove any attributes that depend on only part of the key and place them into a new relation

- We now have two relations:

  R121 = (<u>Item #</u>, <u>*Invoice #*</u>, Qty**)**

  R122 = (Description, Unit Price**)**

  – R12 has been split, removing the attributes that don't depend on the whole primary key

Steps from 1NF to 2NF – Example

2. Then, copy over the part of the primary key that the attributes depend on and make it the *primary key for the new relation*:

   R121 = (<u>Item #</u>, *<u>Invoice #</u>*, Qty)

   R122 = (<u>Item #</u>, Description, Unit Price)

3. That part of the primary key in the original relation is now a *foreign key*, as it refers to the primary key of another relation:

   R121 = (*<u>Item #</u>*, *<u>Invoice #</u>*, Qty)

   R122 = (<u>Item #</u>, Description, Unit Price)

   - Item # in R121 is now a foreign key – italicised
   - Invoice # is also a foreign key, as it relates to the invoice details in R11
   - Together, they are the primary key of R121 – an item in an invoice

Steps from 1NF to 2NF – Example

- Final relations at 2NF:
  - **R11 = (<u>Invoice #</u>,** Order Date, Customer #, Name, Phone, Address**)**
  - **R121 = (*<u>Item #</u>*, *<u>Invoice #</u>*,** Qty**)**
  - **R122 = (<u>Item #</u>,** Description, Unit Price**)**
- This process must be repeated for all partial dependencies in all relations
  - There are no more in this example
- All relations have valid primary keys
- Foreign keys preserve the links between relations
- You can start to see the normalised structure emerging!

Example Summary

R11 = (<u>Invoice #</u>, *Order Date, Customer #, Name, Phone, Address***)** ☺

~~**R12 = (<u>Item #</u>, <u>Invoice #</u>,** *Description, Qty, Unit Price***)**~~

R121 = (<u>Item #</u>, <u>Invoice #</u>, *Qty***)**

R122 = (*Description, Unit Price***)**

R121 = (<u>Item #</u>, <u>Invoice #</u>, *Qty***)**

R122 = (<u>Item #</u>, *Description, Unit Price***)**

2NF:

R11 = (<u>Invoice #</u>, *Order Date, Customer #, Name, Phone, Address***)**

R121 = (<u>Item #</u>, <u>Invoice #</u>, *Qty***)**

R122 = (<u>Item #</u>, *Description, Unit Price***)**

Third Normal Form (3NF)

- **3NF**: A relation is in 3NF if it is in 2NF and *every non-key attribute is mutually independent*

- To achieve 3NF, we need to *resolve all transitive dependencies between non-key attributes*
  - A *transitive dependency* exists where one or more non-key attributes are dependent on another non-key attribute(s), not just on the designated primary key
  - i.e. There is a dependency between non-key attributes in a relation

- ***Note:*** *If no transitive dependencies exist, 3NF is achieved without having to change any of the existing relations*
  - *Some scenarios will also have no partial dependencies, meaning that 2NF is the same as 1NF*

Steps from 2NF to 3NF – Example

- Upon reaching 2NF, the relations in our example are:

  R11 = (<u>Invoice #</u>, Order Date, Customer #, Name, Phone, Address**)**

  R121 = (*<u>Item #</u>*, *<u>Invoice #</u>*, Qty**)**

  R122 = (<u>Item #</u>, Description, Unit Price**)**

  – All attributes in R121 and R122 depend only on the primary key

  – The Name, Phone and Address attributes in R11 rely on Customer #

1. First, *remove any attributes that depend on non-key attribute(s)* and place them into a new relation

- We now have two relations:

  R111 = (<u>Invoice #</u>, Order Date, Customer #**)**

  R112 = (Name, Phone, Address**)**

  – R11 has been split, removing the attributes that depend on non-key attribute(s)

Steps from 2NF to 3NF – Example

2. Then, copy over the attribute(s) that the removed attributes depend on and make it the *primary key for the new relation*:

   R111 = (<u>Invoice #</u>, Order Date, Customer #**)**

   R112 = (<u>Customer #</u>, Name, Phone, Address**)**

3. That attribute(s) in the original relation is now a *foreign key*, as it refers to the primary key of another relation:

   R111 = (<u>Invoice #</u>, Order Date, *Customer #***)**

   R112 = (<u>Customer #</u>, Name, Phone, Address**)**

   – Customer # in R111 is now a foreign key
   – Customer # in R112 is the primary key of the relation

Steps from 2NF to 3NF – Example

- Final relations at 3NF:
  - **R111 = (<u>Invoice #</u>**, Order Date, *Customer #***)**
  - **R112 = (<u>Customer #</u>**, Name, Phone, Address**)**
  - **R121 = (<u>*Item #*</u>, <u>*Invoice #*</u>**, Qty**)**
  - **R122 = (<u>Item #</u>,** Description, Unit Price**)**
- This process must be repeated for all transitive dependencies in all relations
  - There are no more in this example
- All relations have valid primary keys, and each relation contains attributes that directly relate to it
- Foreign keys preserve the links between relations
- The example is now considered normalised!

Example Summary

R121 = (<u>Item #</u>, <u>Invoice #</u>, Qty) ☺

R122 = (<u>Item #</u>, Description, Unit Price) ☺

~~***R11 = (<u>Invoice #</u>, Order Date, Customer #, Name, Phone, Address)***~~

R111 = (<u>Invoice #</u>, Order Date, Customer #)

R112 = (Name, Phone, Address)

R111 = (<u>Invoice #</u>, Order Date, Customer #)

R112 = (<u>Customer #</u>, Name, Phone, Address)

3NF:

R111 = (<u>Invoice #</u>, Order Date, Customer #)

R112 = (<u>Customer #</u>, Name, Phone, Address)

R121 = (<u>Item #</u>, <u>Invoice #</u>, Qty)

R122 = (<u>Item #</u>, Description, Unit Price)

Summary

Summary

- This chapter focuses on modeling data in preparation for implementing a database solution
- "Things" in the problem domain are identified and modelled into data entities that become database tables
- Normalisation can be used to fix anomalies with data and ensure that the correct attributes appear in the correct entities
- There are three common normal forms that are regularly used to normalise data but higher normal forms also exist

Summary

- Entity-relationship diagrams (ERDs) show the information about data entities
- ERDs are often preferred by database analysts and are widely used
- ERDs are not UML diagrams, and an association is called a relationship, multiplicity is called cardinality, and generalization/specialization (inheritance) and whole part relationships are usually not shown

Summary

- Normalisation steps:
  1. Gather the unnormalised data set (covered in Week 1)
  2. Remove the *repeating groups and identify keys* (**1NF**)
  3. Remove all *partial dependencies* (**2NF**)
  4. Remove all *transitive dependencies* (**3NF**)
  5. Name the resultant relations
- *Remember, all of the steps must be completed correctly and in order for the result to be correct*

Questions?