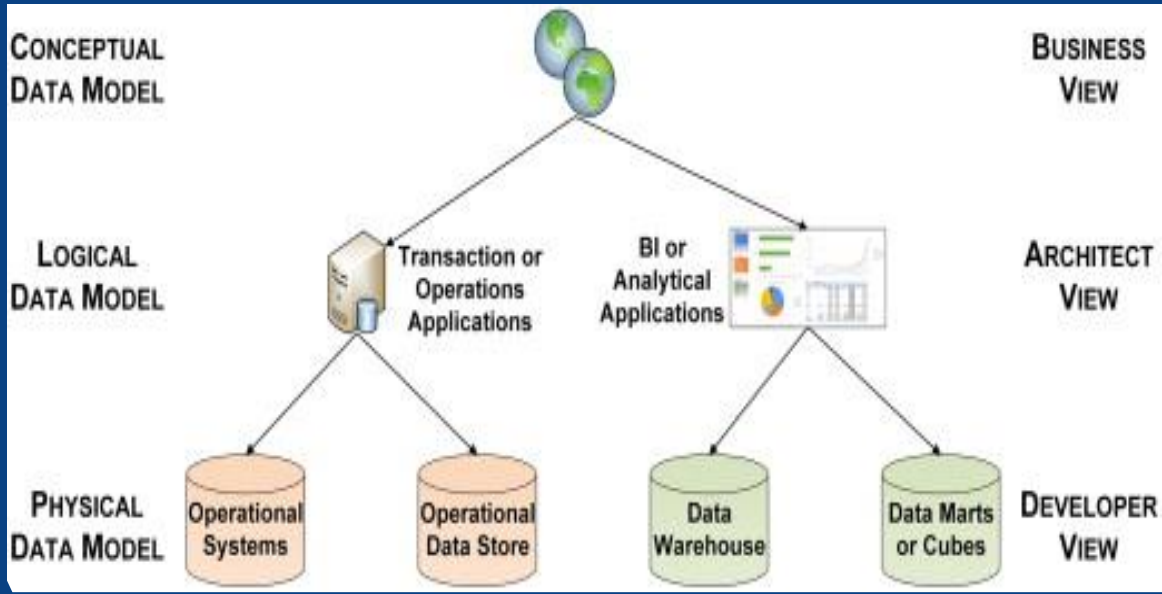
THE UNIVERSITY OF MELBOURNE

Logical & Physical Modelling

Database Systems & Information Modelling INFO90002.



Week 3 – Data Modelling
Dr Tanya Linden
Dr Renata Borovica-Gajic
David Eccles

So far: Introduction to Modelling

Basic ER modeling concepts

- Entities (strong, weak)
- Relationships (non-identifying/weak, identifying/strong)
- Attributes (Key Attributes)

Constraints

- Connectivity Constraints M:M, 1:M, 1:1
- Participation/Cardinality Constraints
  - Mandatory
  - Optional

Good / Bad Entity Selection

- Entity
  - Will have many instances in the database
  - Is composed of many attributes
  - Is something needed for the system to work (something we are trying to model)
- Examples
  - Person: EMPLOYEE, STUDENT, PATIENT
  - Place: STORE, WAREHOUSE, STATE
  - Object: MACHINE, BUILDING, VEHICLE
  - Event: SALE, REGISTRATION, BROADCAST
  - Concept: ACCOUNT, COURSE, ROLE
- An Entity IS NOT
  - A user of the system
  - An output of the system (i.e. a report)

A **Kooyong Road medical practice** needs a database to store details of patients, doctors and appointments.
Questions
1. Is **Kooyong Road medical practice** an entity in the database? No
2. Will a **receptionist** who will be adding patients to the database and recording appointments become an entity in this database? No

Relational Data Model

Data Model allows us to translate real world things into structures that a computer can store

Many models: Relational, ER, O-O, etc.

Relational Model:

- Rows (or Tuples) and Columns (Attributes/fields)
- Primary Keys and Foreign Keys to link Relations

Customer

Customer ID	Surname	First name	Address	Postcode	Credit card
C2045	Smith	Fred	1 John St, Hawthorn	3122	1234234534564567
C2048	Nguyen	Vincent	2/7 Oak Ave, Altona	3018	4554123423457899
C2146	Davis	Liz	32 Lyle St, Toorak	3142	4564564578970022

Order

Order No	Customer ID	Date	Time	Deliv
3224	C2045	1/09/2021	17:35	The b
3228	C2045	3/09/2021	18:42	The b
3236	C2048	3/09/2021	21:05	
3248	C2045	4/09/2021	15:20	The b

Terminology

Every system has its own terminology:

Relational Model terminology	MS Access RDBMS terminology	MySQL, Oracle RDBMS terminology
Relation	Table	Table
Attribute	Field	Column
Tuple or Row	Record	Row

Relational Database: Definitions

Relational database: a set of *relations*.

Relation: made up of 2 parts:

- ***Schema*** : specifies name of relation, plus name and type of each column (attribute).

  Example:

  Pizza(*pID*: string, *pizzaName:* string, *price*: real)

- ***Instance*** : a ***table,*** with rows and columns.

  Number of rows = *cardinality*

  Number of columns/fields = *degree (or arity)*

You can think of a relation as a *set* of *rows* or *tuples*.

- all rows are *distinct*
- *no order* among rows

Example Instance of Pizza Relation

Pizza

Code	Pizza name	Price
P1	Mario's Supreme Pizza	6.95
P2	Vegetarian Pizza	6.95
P3	Hawaiian Pizza	6.95
P4	Hot 'n' spicy Pizza	6.95
B1	Garlic Bread	4.95
B2	Herb Bread	4.95
D1	2 Litre Cola	2.50
D2	2 Litre Lemonade	2.50

Cardinality = 8 (many), degree (arity) = 3, all rows distinct

number of row

number of columns

These two are different

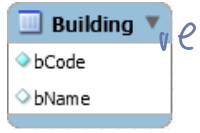
Logical Design: ER to Logical Relation

In logical design **entity** set becomes a **relation.** Attributes become attributes of the relation.

Conceptual Design: → **Logical Relation**: ↔ **Logical ERD**

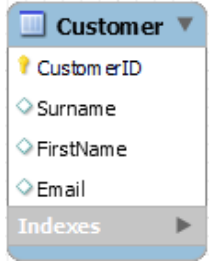
Building	
PK	bCode
	bName

Building (bCode, bName)



Customer	
PK	CustomerID
	Surname
	FirstName
	Email

Customer (customerID, Surname, FirstName, Email)



PK is underlined

ER to Logical to Physical

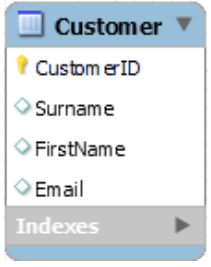
In physical design we specify data types → how much memory space cost how long it take

1. Conceptual Design:

Building	
PK	bCode
	bName

2. Logical Relation:
Building (bCode, bName)

Or **Logical Design Diagram:**

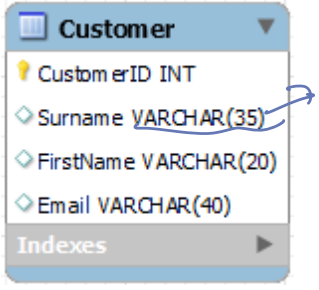


add the data type

3. Physical Relation: (in word)
Building
(bCode CHAR(2), bName
VARCHAR(25)
)

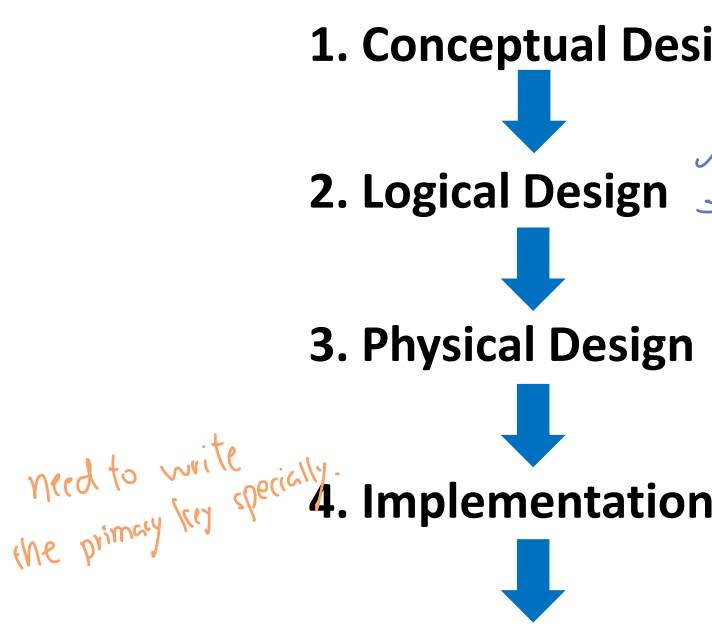
different is its how datatype.

Or **Physical Design Diagram:**



datatype (cost time and space) in computer

The Entire Cycle



no multiple value attribute
no M:M

→ remove multiple value attribute, have M:M

→ need data type

need to write the primary key specially.

covering physical design info SQL and then you start adding data. adding rules to your table.

The Entire Cycle

1. Conceptual Design:

Customer	
PK	CustomerID
	Surname
	FirstName
	Email

2. Logical Relation:

Customer (customerID, Surname, FirstName, Email)

3. Physical Relation:

Customer
(customerID CHAR(11),
Surname VARCHAR(35),
FirstName VARCHAR(20),
Email VARCHAR(40))

data type

4. Implementation:

get

table name

```
CREATE TABLE Customer
    (customerID CHAR(11),
     Surname VARCHAR(35),
     FirstName VARCHAR(20),
     Email VARCHAR(40),
     PRIMARY KEY (customerID))
```

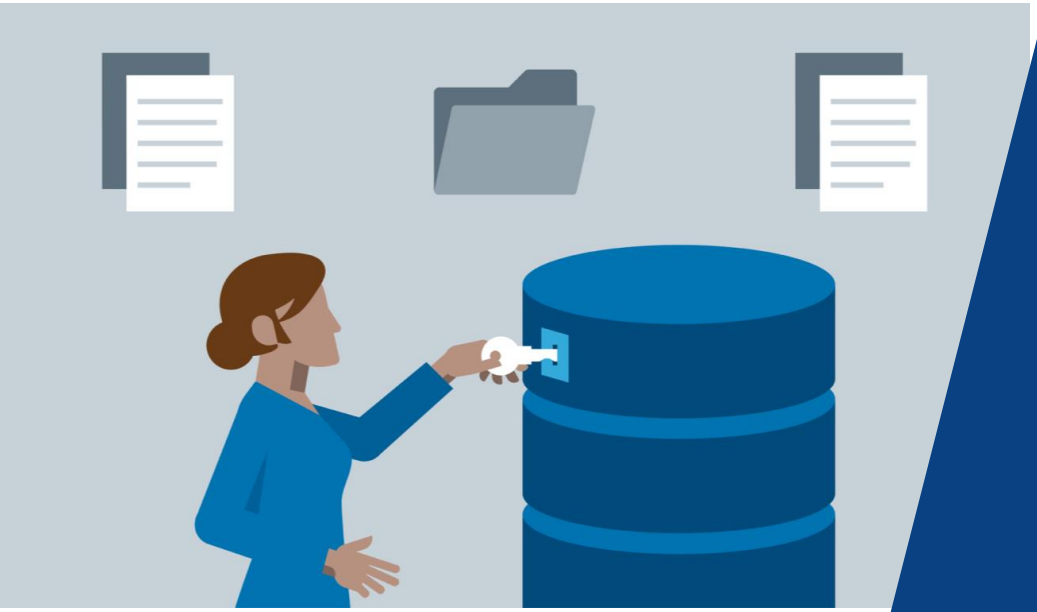
specific classify the underline

5. Instance:

CUSTOMER

CustomerID	Surname	FirstName	Email
0983763423	Ellis	John	ellisj@xyz.com
9384392483	Lee	Jane	jane@aa.com
3743923483	Smith	Jill	js@gmail.com

THE UNIVERSITY OF MELBOURNE



Keys and Integrity Constraints

Keys

In human logic: order is weak entity need the foreign key

In logic; order can be strong entity, some store not need the customer name. order number is it primary key.

Keys are a way to associate tuples in different relations

Keys are one form of **integrity constraint (IC)**

Example: *Only customers can place orders.*

Each Foreign Key value in the Order table references a Primary Key value in the Customer table

Customer

Customer ID	Surname	First name	Address	Postcode	Credit card
C2045	Smith	Fred	1 John St, Hawthorn	3122	1234234534564567
C2048	Nguyen	Vincent	2/7 Oak Ave, Altona	3018	4554123423457899
C2146	Davis	Liz	32 Lyle St, Toorak	3142	4564564578970022

PRIMARY Key

Is foreign key for order

Order

Order No	Customer ID	Date	Time	Deliver
3224	C2045	1/09/2021	17:35	The be
3228	C2045	3/09/2021	18:42	The be
3236	C2048	3/09/2021	21:05	
3248	C2045	4/09/2021	15:20	The be

FOREIGN Key

could be one of business rule create the order number.

Superkey

A set of fields is a ***superkey*** if no two distinct tuples can have same values in all key fields

The set of **all** attributes is always a superkey.
Example {CustomerID, Surname, Firstname, Address, Postcode, CreditCard}

Other superkeys are combination of CustomerID and any other fields, e.g.

- {CustomerID, Surname, Firstname, Address, Postcode}
- {CustomerID, Surname, Firstname, Address}
- {CustomerID, Surname, Firstname}, …

A set of fields is a ***key*** for a relation if it is a superkey and no subset of the fields is a super key (minimal subset, can't be made smaller)

A minimal superkey is a superkey that can't be reduced to a simpler superkey by removing an attribute

Customer ID	Surname	First name	Address	Postcode	Credit card
C2045	Smith	Fred	10 John St, Hawthorn	3122	1234234534564567
C2048	Nguyen	Vincent	3/1 Oak Ave, Altona	3018	4554123423457899
C2146	Davis	Liz	32 Lyle St, Toorak	3142	4564564578970022

Primary Keys

We can't not choose the attributes that will change to be primary key.

Out of all keys *one* is chosen to be the ***primary key*** of the relation.

Other keys are called ***candidate*** keys. sid and login

Each *relation* has a ***primary key***.

↳ choose the attribute that will not be changed

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@cs	18	3.2
53650	Smith	smith@math	19	3.8

primary key — not primary key — this one will change — change

Your turn:

1. Is *sid* a key for Student? yes
2. What about *name*? No (many same name)
3. Is the set {*sid, gpa*} a superkey? yes Is the set {*sid, gpa*} a primary key? Yes
4. Find a primary key from this set {*sid, login*} sid

can't change

student id is primary key

Because any combination of fields which define the records unique is a super key.

Selecting the Primary Key

Superkeys

Candidate Keys

Primary Key

- Superkey – a set of fields that contains the key
- Candidate keys are all the possible key combinations that could be the Primary Key
- Of all candidate keys the database designer *identifies* the primary key. The primary key is the *fewest number of columns* that can uniquely identify a key. It should be an attribute that will not change throughout the life of the entity.
- N.B.* Not all relations will have a candidate key. In cases when there is no candidate key, the database designer will add a *surrogate key*

*N.B. means Nota Bena, Latin for "Note well" or in simple terms "This information is important!"

Surrogate Keys

A surrogate key is

- A key that has no real world / business meaning
- Is usually numeric
- Is often a sequential number supplied by the RDBMS
  (e.g. AUTO_INCREMENT in MySQL, AutoNumber in MS Access)

Many databases around the world have been created with all their tables using surrogate keys.

When modelling business requirements

- You **speak** to **clients**
- You use terms **applicable** to **their** business
- You **do not invent** terms / fields that **do not match** their business
- You use **Natural Keys**

Surrogate vs Natural Keys

Example:

If a small college teaches 20 subjects, that business **may not** have/use subject codes. When identifying a subject, they simply use **natural keys** such as Subject Name.

While Subject Name may seem obviously **inadequate** for a large database, it may be sufficient for Modelling requirements.

Forcing a term such as Subject Code into the conversation may **confuse** clients.

Question: college management decided that staff work email or mobile phones are sufficiently unique to use as identifiers for staff members. Is this a good design decision for a real database? No, email and mobile phones number will change

Modelling with Natural Keys

Modelling typically **uses Natural Keys**

Modelling is sometimes the **end of the process.**

- The **ER Model** that is produced may be the **final product**
  - There is **no database**
  - Both the **client** and the **modeler learn** about the business and its requirements via the modelling process.

If a Database is required:

- Database **implementers** can choose to **add surrogate keys**.
  - When adding a surrogate key – **you do not lose any data.**
  - The **natural key** data is **not removed.** It's just not chosen to be a primary key.

Foreign Keys and Referential Integrity

Foreign key : A set of fields in one relation that is used to 'refer' to a tuple in another relation. The foreign key must correspond to the primary key of the other relation, i.e. the PK must exist!

If all foreign key constraints are enforced in a DBMS, we say a ***referential integrity*** is achieved.

Foreign Keys in SQL

Example: *Only customers listed in the Customer relation should be allowed to place Orders.*

In Order *customerID* is a foreign key referring to Customer table

schema

```
CREATE TABLE Order
(orderNo CHAR(5),
 customerID CHAR(8) NOT NULL,
 ...,
 PRIMARY KEY (orderNo),
 FOREIGN KEY (customerID) REFERENCES Customer (customerID))
```

Customer

Customer ID	Surname	First name	Address	Postcode	Credit card
C2045	Smith	Fred	1 John St, Hawthorn	3122	1234234534564567
C2048	Nguyen	Vincent	2/7 Oak Ave, Altona	3018	4554123423457899
C2146	Davis	Liz	32 Lyle St, Toorak	3142	4564564578970022

Order

Order No	Customer ID	Date	Time	Deliver
3224	C2045	1/09/2021	17:35	The be
3228	C2045	3/09/2021	18:42	The be
3236	C2048	3/09/2021	21:05	
3248	C2045	4/09/2021	15:20	The be

Enforcing Referential Integrity

Consider *Customer* and *Order*:

- *customerNo* in Order is a foreign key that references Customer.

What should be done if an Order tuple with a non-existent customerNo is inserted? (*Reject it!*)

What should be done if a Customer tuple is deleted?

- Also delete all Order tuples that refer to it?
- Disallow deletion of a Customer tuple that is referred to?
- Set customerNo in Order tuples that refer to it to a *default customerNo*?
- (In SQL, also: Set customerNo in Order tuples that refer to it to a special value *null*, denoting *'unknown'* or *'inapplicable'*)

Note: Similar issues arise if primary key of Customer tuple is updated.

Integrity Constraints (ICs)

IC: condition that must be true for *any* instance of the database; e.g., *domain constraints*.

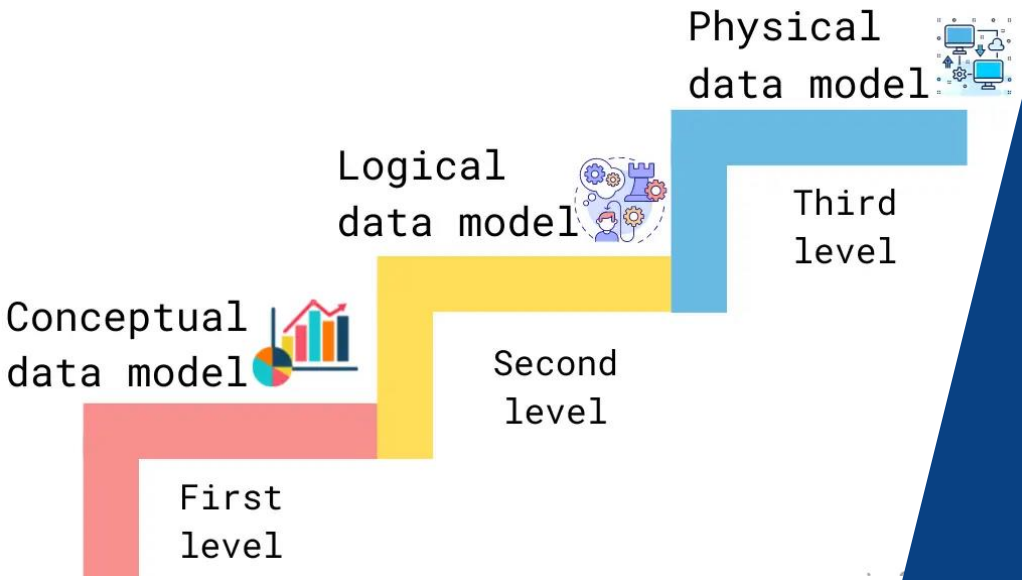
- ICs are specified when schema is defined.
- ICs are checked when relations are modified.

A ***legal*** instance of a relation is one that satisfies all specified ICs.

- DBMS should not allow illegal instances.

This is also known as **Schema on Write**

- The schema table structure is known in advance of the data to be inserted into it



From Conceptual to Logical and to Physical

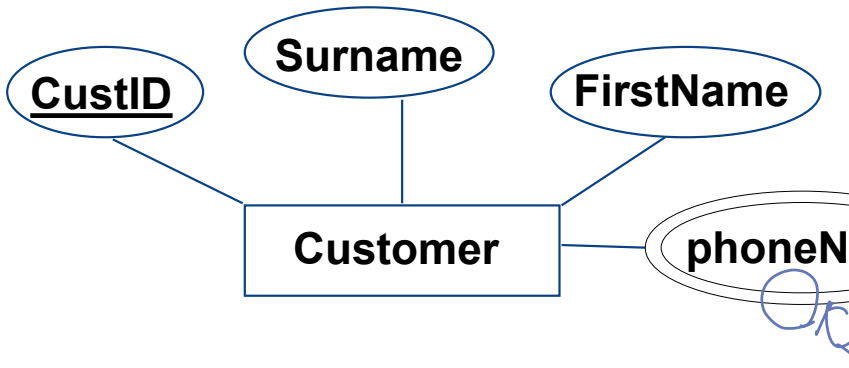
Multi-valued attributes in logical design

Multi-valued attributes need to be unpacked (flattened) when converting to logical design.

Example:

For a customer a bank needs to capture home phone number, work phone number and mobile.

Conceptual Design:
(Chen notation)

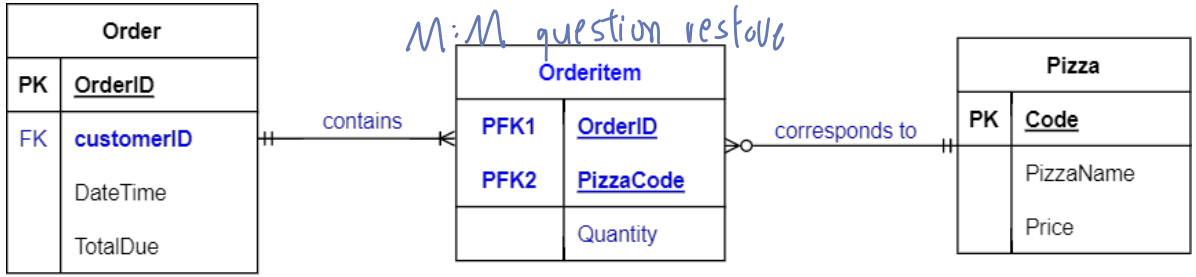


Logical Relation:

Customer (CustID,
Surname,
FirstName,
home_num,
work_num,
mobile)

ER to Logical Design

Conceptual/Logical Design Diagram:



Logical Relations:

Pizza (<u>Code</u>, PizzaName, Price)

Order (<u>OrderID</u>, *CustomerID*, DateTime, TotalDue)

OrderItem (<u>*OrderID*</u>, <u>*PizzaCode*</u>, Quantity)

Keys from connecting entities become PFK

Note, PFK name does **not** have to match PK name of the parent entity

Note:
Underline = PK,
Italic = FK,
Underline + Italic = PFK

Logical to Physical Design

Logical Relations:

Pizza (Code, PizzaName, Price)

Order (OrderID, *CustomerID*, DateTime, TotalDue)

OrderItem (*OrderID*, *PizzaCode*, Quantity)

add datatype

Note:
Underline = PK,
Italic = FK,
Underline + Italic = PFK

Physical Relations:

```
Pizza (
  Code CHAR(6),
  PizzaName VARCHAR(15),
  Price DECIMAL(5,2)
)
```

```
Order (
  OrderID  INTEGER,
  CustomerID INTEGER,
  OrderDateTime  DATETIME,
  TotalDue DECIMAL(6,2)
)
```

data type

```
OrderItem (
  OrderID  INTEGER,
  PizzaCode CHAR(6),
  Quantity INTEGER
)
```

Implementation (CREATE TABLE)

Note:
Underline = PK,
Italic = FK,
Underline + Italic = PFK

Physical Relations:

Pizza (
  Code CHAR(6),
  PizzaName VARCHAR(15),
  Price DECIMAL(5,2)
)

Order (
  OrderID INTEGER,
  CustomerID INTEGER,
  OrderDateTime DATETIME,
  TotalDue DECIMAL(6,2)
)

OrderItem (
  OrderID INTEGER,
  PizzaCode CHAR(6),
  Quantity INTEGER
)

Implementation:

(like create table specify primary key and foreign key and reference s.

```
Pizza (
  Code CHAR(6),
  PizzaName VARCHAR(15),
  Price DECIMAL(5,2),
  PRIMARY KEY (Code)
)
```

```
Order (
  OrderID INTEGER,
  CustomerID INTEGER,
  OrderDateTime DATETIME,
  TotalDue DECIMAL(6,2),
  PRIMARY KEY (OrderID),
  FOREIGN KEY (CustomerID)
        REFERENCES Customer
)
```

```
OrderItem (
  OrderID INTEGER,
  PizzaCode CHAR(6),
  Quantity INTEGER,
  PRIMARY KEY (OrderID, PizzaCode),
  FOREIGN KEY (OrderID)
        REFERENCES Order,
  FOREIGN KEY (PizzaCode)
        REFERENCES Pizza(Code)
)
```

PFK have both

Sample Instances put the value in

Order

Order No	Customer ID	Date	Time	Delivery Instructions	Total
3224	C2045	1/09/2021	17:35	The bell does not work,	59.55
3228	C2045	3/09/2021	18:42	The bell does not work,	28.35
3236	C2048	3/09/2021	21:05		51.45
3248	C2045	4/09/2021	15:20	The bell does not work,	13.90

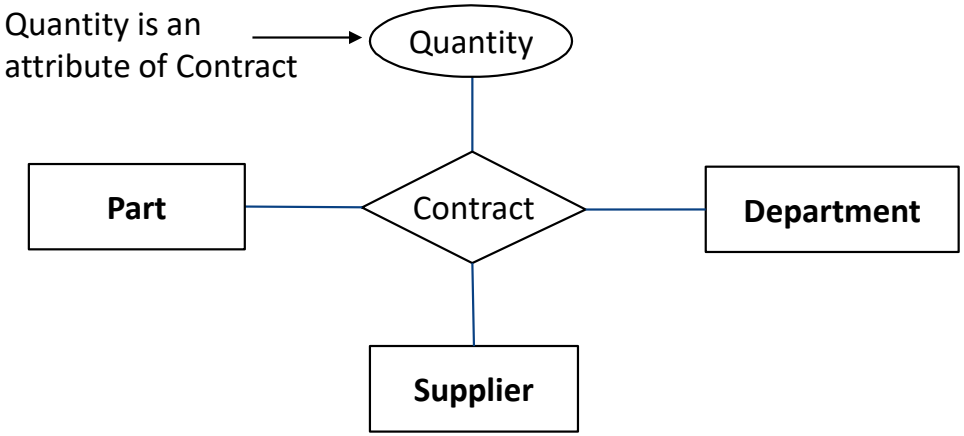
Pizza

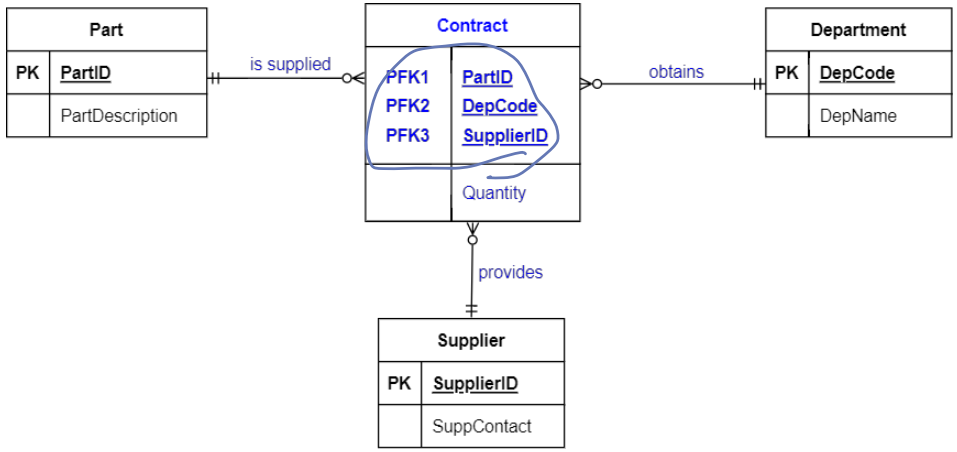
Code	Pizza name	Price
P1	Mario's Supreme Pizza	6.95
P2	Vegetarian Pizza	6.95
P3	Hawaiian Pizza	6.95
P4	Hot 'n' spicy Pizza	6.95
B1	Garlic Bread	4.95
B2	Herb Bread	4.95
D1	2 Litre Cola	2.50
D2	2 Litre Lemonade	2.50

OrderItem

Order No	Line Item	Quantity
3224	P1	2
3224	P3	1
3228	P4	1
3228	B1	4
3228	D1	2

Ternary Relationships to Logical Relation





Logical Relation:

Contract (
SupplierID, PFK
PartID,
DepCode,
Quantity
)

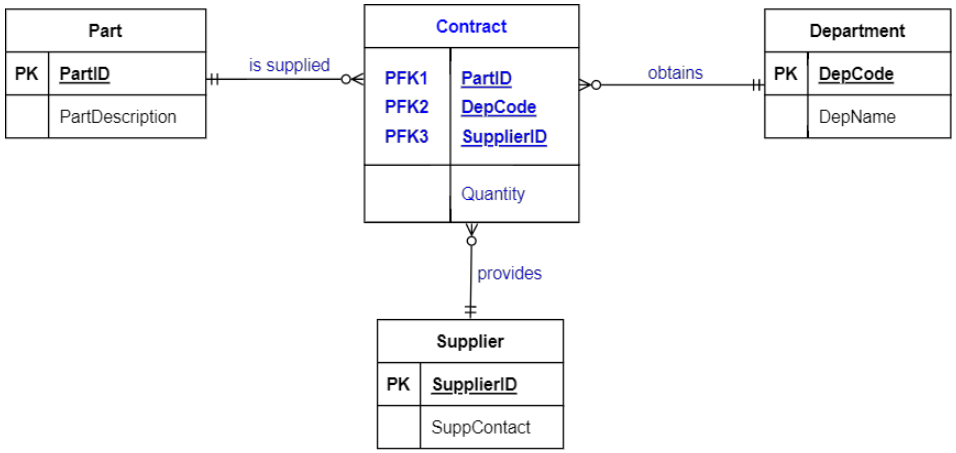
In translating an n-ary relationship set to a logical relation, attributes of the relation must include:

- PKs for each participating entity set (as foreign keys). This set of attributes is likely to form a *superkey* for the relation.
- All descriptive attributes.

ER to Logical to Implementation Example

Logical Relation:

Contract (
SupplierID,
PartID,
DepCode,
Quantity
)

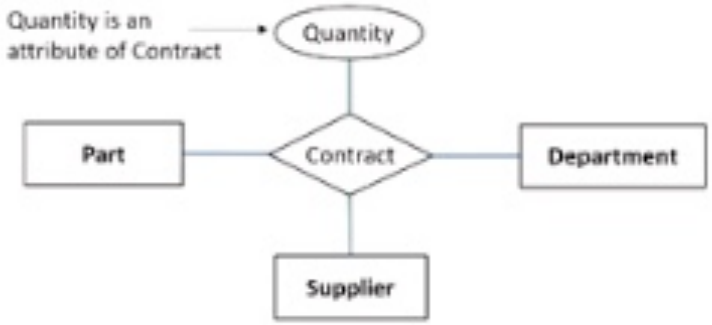


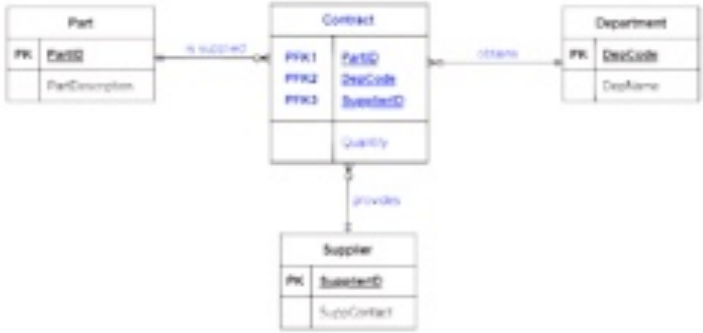
Implementation:

```
CREATE TABLE Contract (
        SupplierID  INTEGER,
        PartID  INTEGER,
        DepCode  CHAR(3),
        PRIMARY KEY (SupplierID, PartID, DepCode),
        FOREIGN KEY (SupplierID)    REFERENCES Supplier,
        FOREIGN KEY (PartID)        REFERENCES Part(PartID),
        FOREIGN KEY (DepCode)       REFERENCES Department)
```

for individual primary key and FK, references.

Ternary Relationships to Logical Relation





Logical Relation:

Contract (
SupplierID,
PartID,
DepCode,
Quantity
)

In translating an n-ary relationship set to a logical relation, attributes of the relation must include:

- PKs for each participating entity set (as foreign keys). This set of attributes is likely to form a *superkey* for the relation.
- All descriptive attributes.

ER to Logical to Implementation Example

Logical Relation:

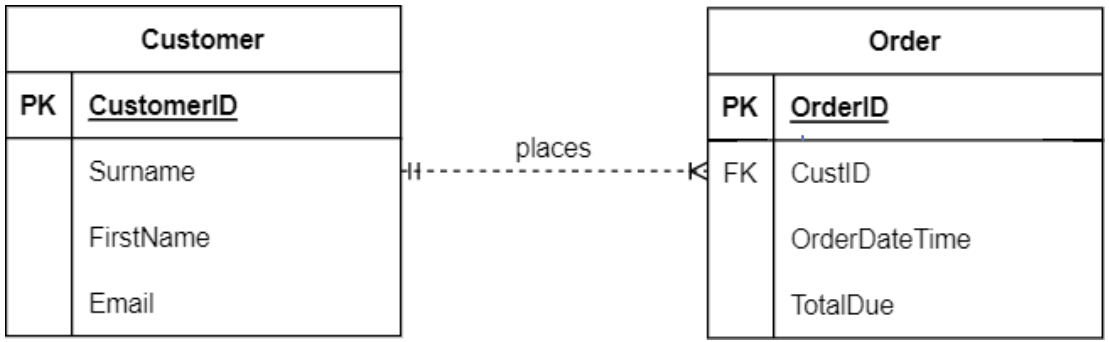
Contract (
SupplierID,
PartID,
DepCode,
Quantity
)

Implementation:

```
CREATE TABLE Contract (
        SupplierID  INTEGER,
        PartID  INTEGER,
        DepCode  CHAR(3),
        PRIMARY KEY (SupplierID, PartID, DepCode),
        FOREIGN KEY (SupplierID)    REFERENCES Supplier,
        FOREIGN KEY (PartID)        REFERENCES Part(PartID),
        FOREIGN KEY (DepCode)       REFERENCES Department)
```

Connectivity Constraints: Logical design

Each order belongs to one Customer. A customer can place many orders.



Logical Design:

Customer (<u>CustID</u>, Surname, FirstName, Email)

Order (<u>orderID</u>, ordDateTime, TotalDue)

Placement (<u>*CustID*</u>, <u>*orderID*</u>)

vs.

Customer (<u>CustID</u>, Surname, FirstName, Email)

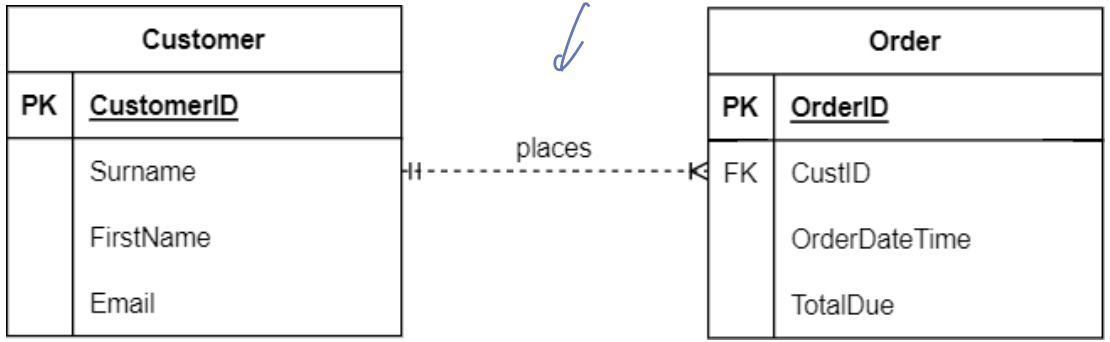
Order (<u>orderID</u>, *CustID*, ordDateTime, TotalDue)

Legend:
Underline = PK,
Italic = FK,
Underline + Italic = PFK

Connectivity Constraints in SQL

Implementation:

Don't need associate entity




```
CREATE TABLE Customer
   (CustID CHAR(11),
    Surname VARCHAR(35),
    FirstName VARCHAR(25),
    Email VARCHAR(30),
    PRIMARY KEY  (CustID))
```

vs.

```
CREATE TABLE  Placement(
 CustID  CHAR(11),
 orderID  INTEGER,
 PRIMARY KEY  (CustID, orderID),
 FOREIGN KEY (CustID) REFERENCES Customer,
 FOREIGN KEY (orderID) REFERENCES Order)
```

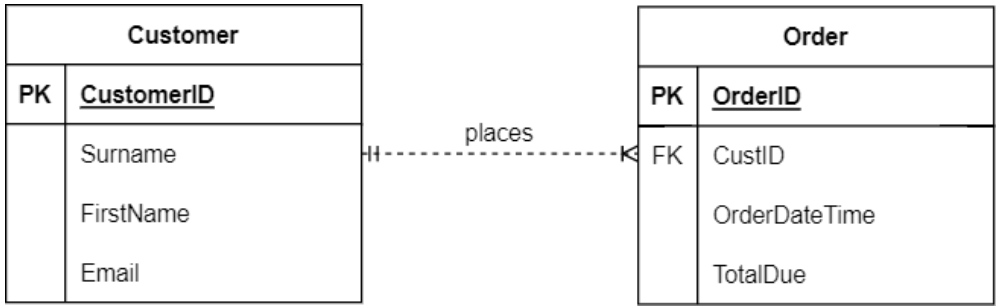
```
CREATE TABLE Order
  (orderID INTEGER,
   CustID CHAR(11),
   orderDateTime DATETIME,
   TotalDue DECIMAL(6,2),
   PRIMARY KEY  (orderID),
   FOREIGN KEY (CustID) REFERENCES
               Customer)
```

Which one is better?

Connectivity Constraint rule

RULE: Primary key from the ***one*** side becomes a foreign key on the ***many*** side

This is the way to ensure 1:M




```
CREATE TABLE Order
  (orderID INTEGER,
   CustID CHAR(11),
   orderDateTime DATETIME,
   TotalDue DECIMAL(6,2),
   PRIMARY KEY  (orderID),
   FOREIGN KEY (CustID) REFERENCES Customer)
```

Each order belongs to a *single* customer

Participation/Cardinality Constraints in SQL

We specify mandatory participation with key words NOT NULL

- NOT NULL = this field cannot be empty

```
CREATE TABLE Order
  (orderID INTEGER,
   CustID CHAR(11)  NOT NULL,
   orderDateTime DATETIME,
   TotalDue DECIMAL(6,2),
   PRIMARY KEY  (orderID),
   FOREIGN KEY (CustID) REFERENCES Customer
            ON DELETE NO ACTION)
```

or it will delete whole

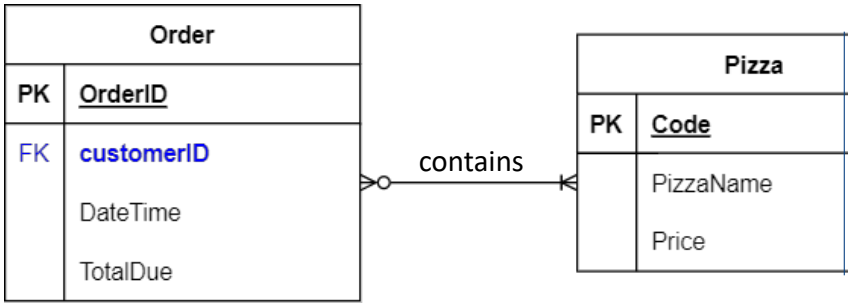
Note, **NO ACTION specification** (vs CASCADE)

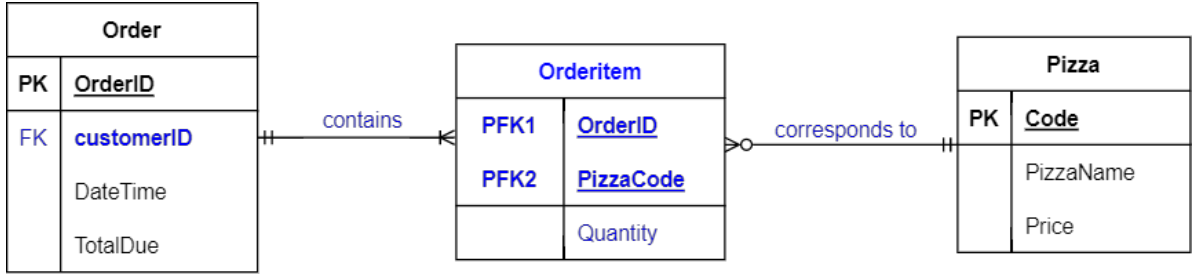
It is the default and need not be explicitly specified; it ensures that a Customer tuple cannot be deleted while it is pointed to by an Order tuple. If we wish to delete such a Customer tuple, we must first change the Customer FK value (e.g. many years down the track it could be some default value).

Review: Weak Entities

A ***weak entity*** can be identified uniquely only by considering the primary key of another (*owner*) entity.

Order and Pizza were M:M with the relationship "Contains" which is mapped onto a weak entity OrderItem when M:M is resolved





In this case when an owner entity is deleted, we want all owned weak entities to be deleted.

Translating Weak Entity Sets

Weak entity set and identifying relationship set are translated into a single table.

- Contains is not the best entity name when we convert a relationship into an entity
- When the owner entity is deleted, all owned weak entities must also be deleted.

In weak entity

Logical Design:

OrderItem (*pizzaID*, qty, *orderNo*)

Note:
Underline = PK,
Italic = FK,
Underline + Italic = PFK

Implementation:

```
CREATE TABLE  OrderItem (
   pizzaCode  CHAR(3) NOT NULL,
   qty  INTEGER,
   orderNo  INTEGER NOT NULL,
   PRIMARY KEY  (pizzaCode, orderNo),
   FOREIGN KEY  (orderNo) REFERENCES Order(orderNo) ON DELETE CASCADE,
   FOREIGN KEY  (pizzaCode) REFERENCES Pizza ON DELETE CASCADE)
```

What's examinable*

Be able to model a case study from conceptual to instance and all stages in between (conceptual, logical, physical, implementation and instance)

Translate conceptual (ER) into logical & physical design

Physical Crow's foot with MySQL Workbench

Understand integrity constraints

Use DDL of SQL to create tables with constraints

* All material is examinable – these are the suggested key skills you would need to demonstrate in an exam scenario

THE UNIVERSITY OF MELBOURNE

Thank you