

Classical Database Development Methodology

Classical Database Development Methodology

- Area of Application
- Perspective
- Work-Processes
- Guidelines for Work-Processes in the development of the application

Area of Application:

- Development of medium to large size data intensive applications
- Data intensive:
 - lots of data
 - little processing
 - insertions, deletions, updates,
 - queries
- What is medium to large?
- Small is:
 - well-defined project
 - short development time
 - no long-term maintenance
 - few people; little turnover
 - no critical resources
 - small risk of failure
 - small cost of failure
- Why only medium to large?
 - the methodology is an insurance policy
 - cost of using methodology is high

Perspective:

- Business process is well-designed
- Documents are known
- Tasks are known
- System boundary is known
- One database schema unifying all views can be designed
 - difficult: interests, goals, power, politics
 - problems with the methodology?
 - problems with the organization?
 - or-gan-i-za-tion: “an entity created to pursue a shared set of goals”

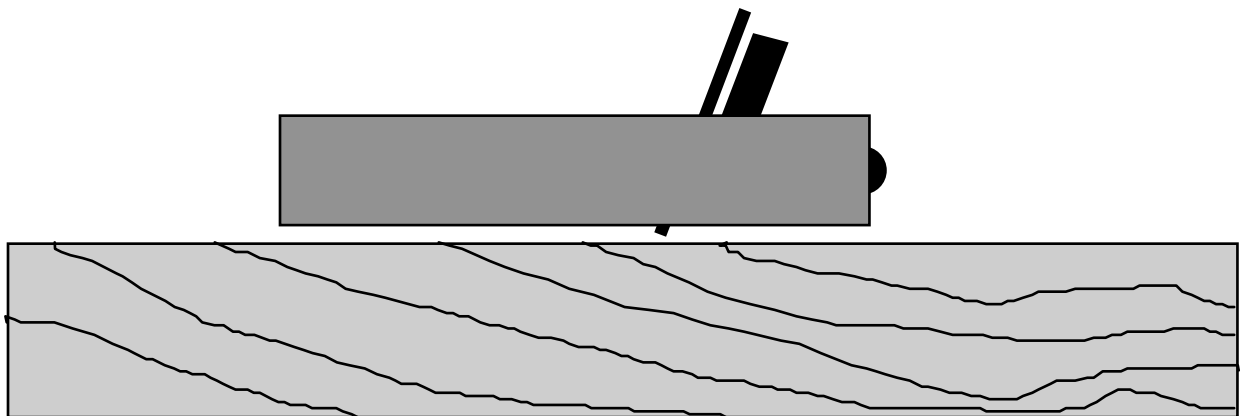
Work-processes:

- Business process (re-)design
- Analysis
- Specification
- Design
- Implementation
- Testing
- Operation
- Maintenance



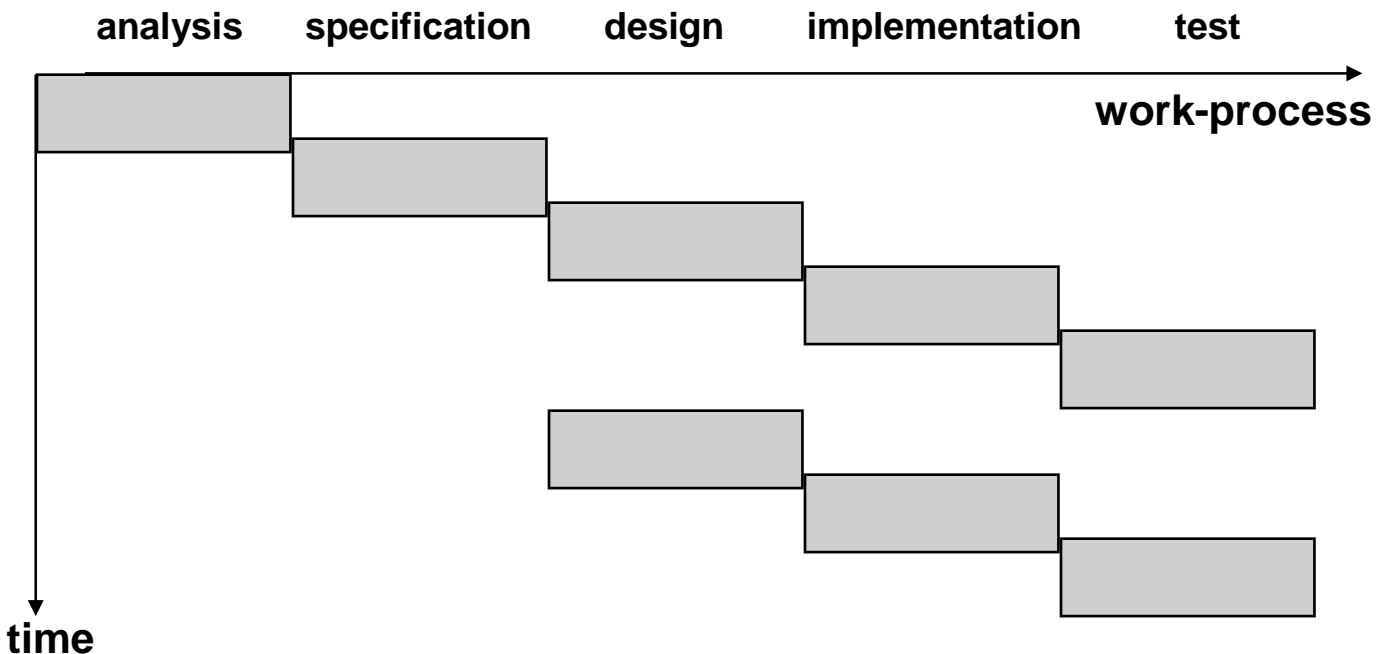
Guidelines for work-processes:

- **Purpose:** what we do
- **Input:** what we start with
- **Output:** what we end with
- **Tool:** what we use
- **Technique:** how we use it
- **Organization:** who does what

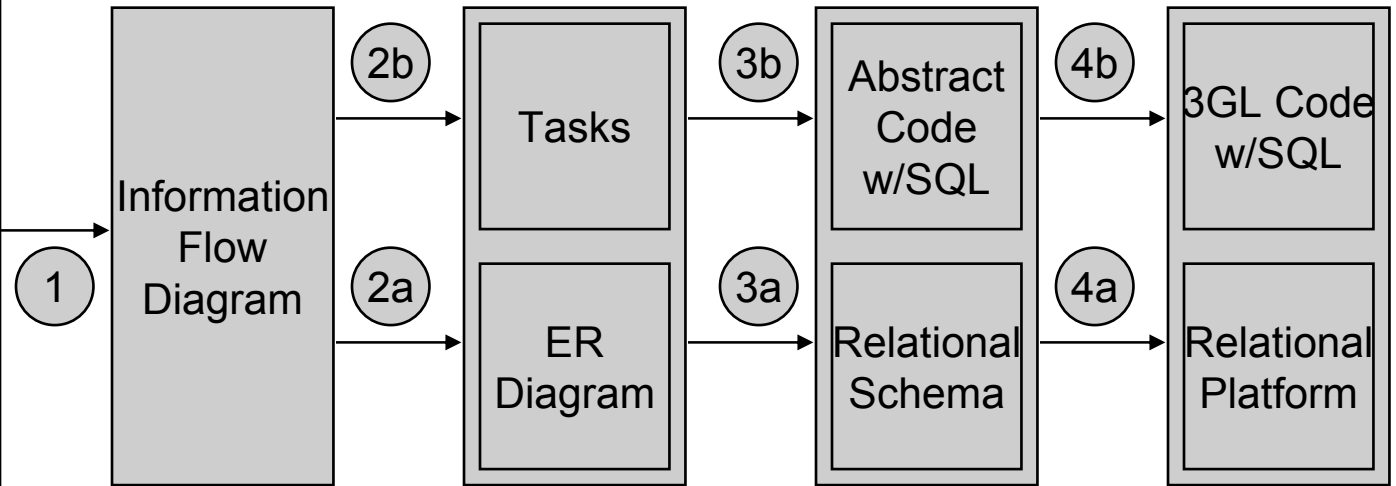


Time and Management

- waterfall model; this is **not** prototyping
- iteration necessary
- work vs. time vs. people
- estimating resources is **very** difficult
- ACM's ethics code



Overview of the Methodology



- ① Analysis
- ② Specification
- ③ Design
- ④ Implementation

Analysis

Analysis

Purpose:

- analyze documents and tasks; determine system requirements

Input:

- descriptions of documents and tasks; scenarios; usage statistics; plans for the future system; relevant laws, constraints, and policies

Output:

- Information Flow Diagram (IFD) modeling external I/O documents, internal I/O documents, tasks, and system boundary.

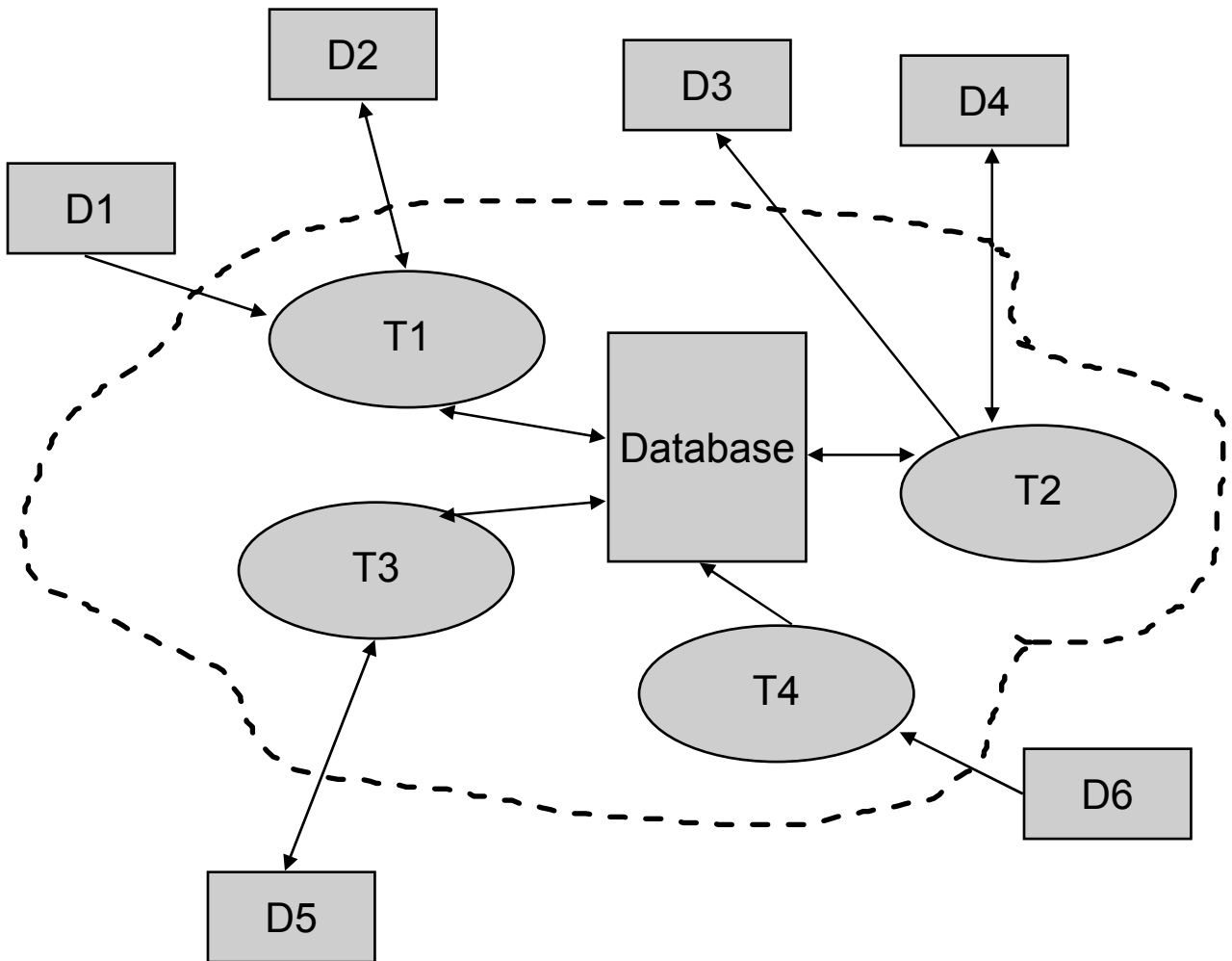
Techniques:

- interviews with people at all levels of the enterprise
- analysis of documents, scenarios, tasks
- reviews of short and long-term plans, manuals, files, and forms
- work from outside in
- abstraction

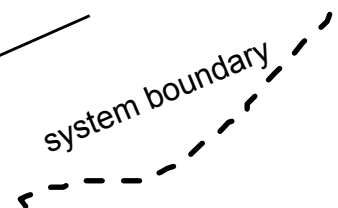
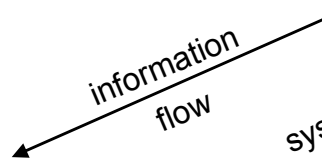
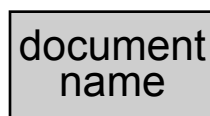
Tools:

- Information Flow Diagrams

Information Flow Diagram



- information flow; **not** control flow
- **never** connect two documents
- **never** connect two tasks



Example

Example External Documents

Flight-Schedule

AIRLINE

From City

To City; Flt#; Dtime; Atime; Weekdays; miles; price

<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
-	-	-	-	-	-	-

Airports

Airport Code Name City State

<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
-	-	-	-

Airplanes

Plane# Plane type Total #seats

<input type="text"/>	<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>	<input type="text"/>
-	-	-

Ticket

Airline

Ticket#

Customer Name

From To Flt# Date Dtime Atime

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

Price

Passenger List

Date

Flt#

Airline

Customer Name Seat#

-	-
-	-

Boarding Pass

Airline

seat#

Customer Name

From To Flt# Date Dtime Atime

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

Check-In/Seat selection

Ticket#

Seat

Assign Flight

Date: (yy-mm-dd)

Flt#:

Plane#

Create Flight Instance

Date: (yy-mm-dd)

Flt#:

Example External Documents

Inquiry Entry

Inquiry

From:

To:

Date:

Reservation

Flight No: Flight Date:

Customer Info

First Name:

Middle Name:

Last Name:

Street Addr:

City:

State: Zip:

Phone No:

Inquiry Result

Number of Leg(s):

Example Scenarios

- Staff **enters airport** information.
- Staff **enters airplane** information.
- Staff **enters flight schedule** information.
- Staff **creates instance of scheduled flight**.
- Staff **assigns airplane** to flight instance.
- Customer **inquires** about direct, 1-leg, or multi-leg flights from departure airport to arrival airport on a desired travel date. Inquiry is answered.
- Customer provides flight number, travel date, and customer information and **makes a reservation**. Ticket is printed. Or, customer **cancels an existing reservation**.
- Customer **checks in** and selects seat on a flight instance he or she has reservation for. Boarding pass is issued.

Example Tasks

- Answer Inquiry
- Make Reservation/Cancellation
- Enter Flight-Schedule
- Create Flight Instance
- Enter Airports
- Enter Planes
- Assign Planes
- Process Check-In

Example Statistics

The Airline Reservation System supports 3 airlines..

Each airline has about 100 planes.

Each plane departs an average of 4 times per day.

There are 6 hubs each of which is completely connected to the others with 1 flight per hour 18 hours per day.

Each of the 6 hubs is connected to about 6 non-hub cities with 1 flight every 2 hours 18 hours per day.

About 30% of all reservations are cancelled.

Planes are over-booked by approximately 10%.

Each plane has 250 seats and is on the average filled 77%.

About 30,000 inquiries per day do not result in reservations.

About 90% of all inquiries deal with direct flights only.

About 10% of all inquiries deal with direct and 2-leg flights.

About 1% of all inquiries deal with n-leg fights, $n > 2$.

About 5% of all reservations are made by new customers.

Customers fly on the average 1 time per month.

At any given time, about half of the flights scheduled over the next 6 months are instantiated.

At any given time, about half of the reservations for the customers who will travel the following 30 days are in the database.

Specification

Specification

Purpose:

- create detailed specification of internal documents and tasks from the IFD

Input:

- IFD, usage statistics, and other information gathered during the analysis

Output:

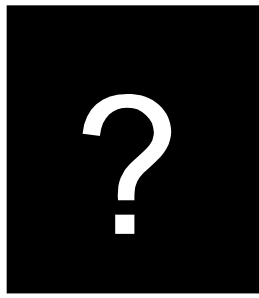
- ER-Diagram, Data Representation, Constraints, Task Decomposition, Task Forms, Task Statistics

Techniques:

- data modeling
- top-down decomposition of tasks until their specification is sufficiently detailed to allow a programmer to implement them
- task decomposition may result in tasks replacing the original task or in subtasks controlled by the original task

Tools:

- ER-Model; Task Forms



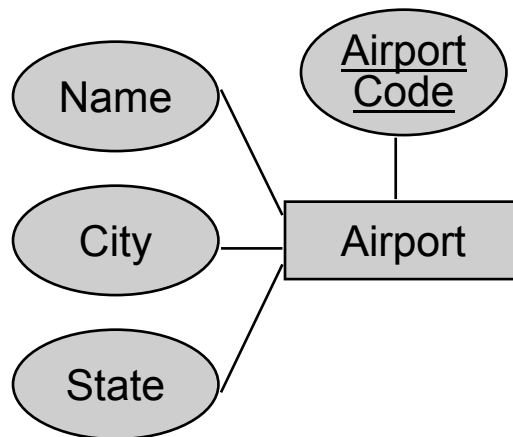
What goes into the database?

What comes out of the database?

- Everything in the database must come from somewhere
- Everything on the input documents must go somewhere
- Everything in the database must be used for something
- Everything on the output documents must come from somewhere

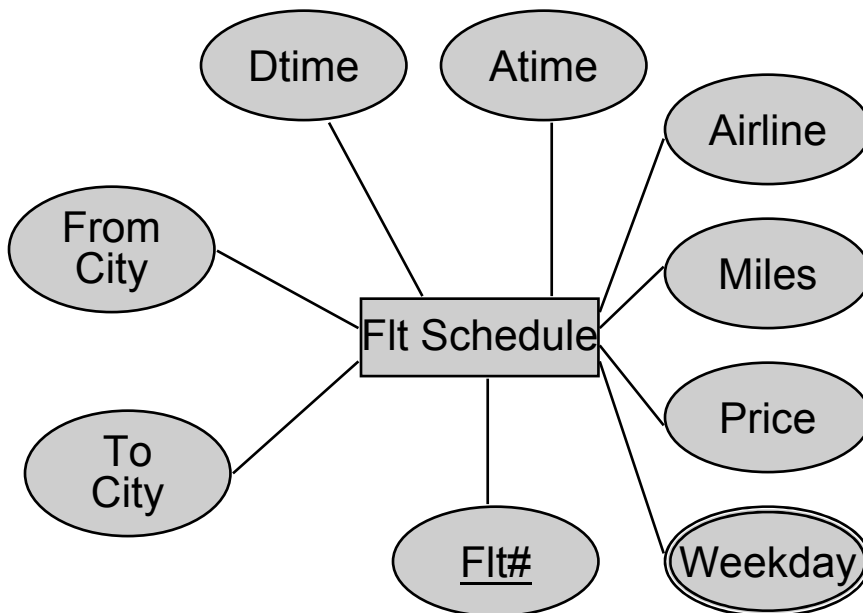
Example ER-Diagram

Airports			
Airport Code	Name	City	State
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
-	-	-	-
-	-	-	-

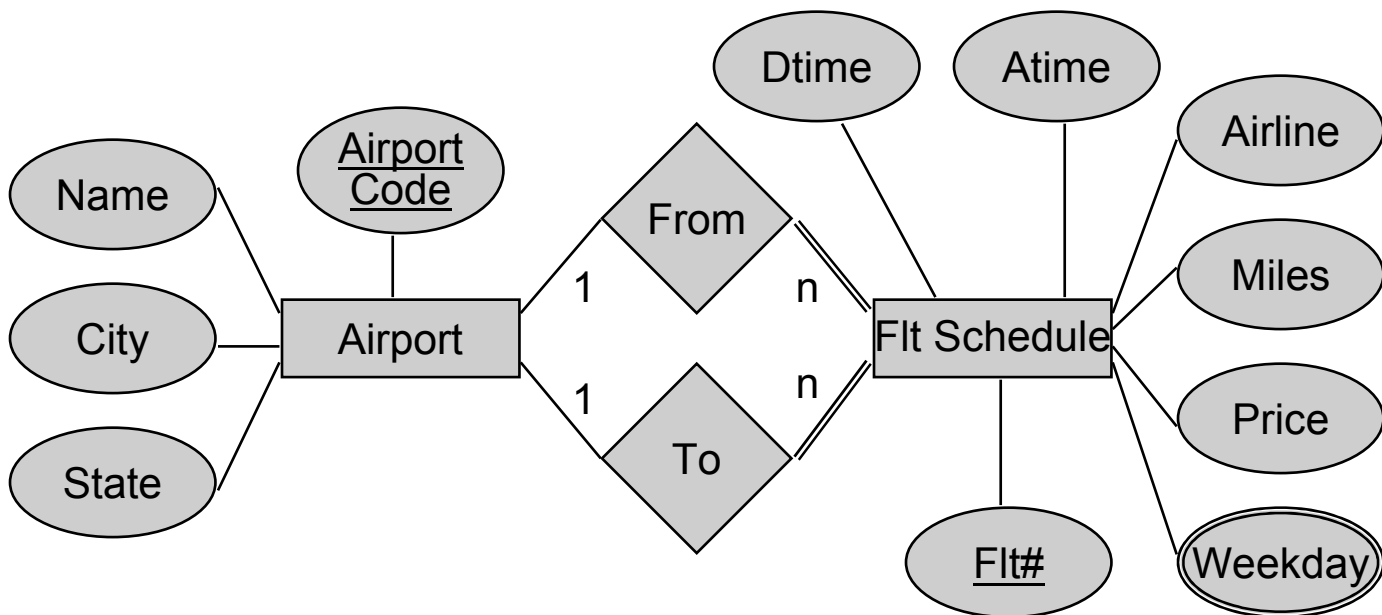
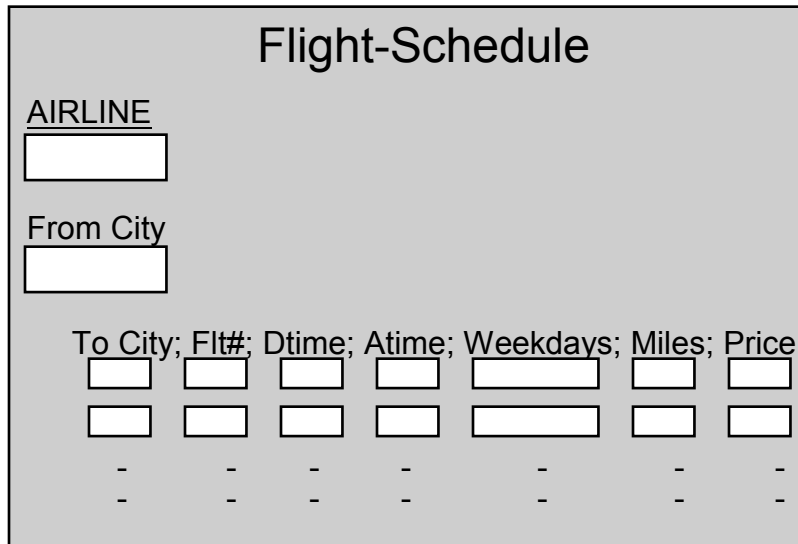


Example ER-Diagram

Flight-Schedule						
<u>AIRLINE</u>						
From City						
To City; Flt#; Dtime; Atime; Weekdays; Miles; Price						
-	-	-	-	-	-	-



Example ER-Diagram (integrate)

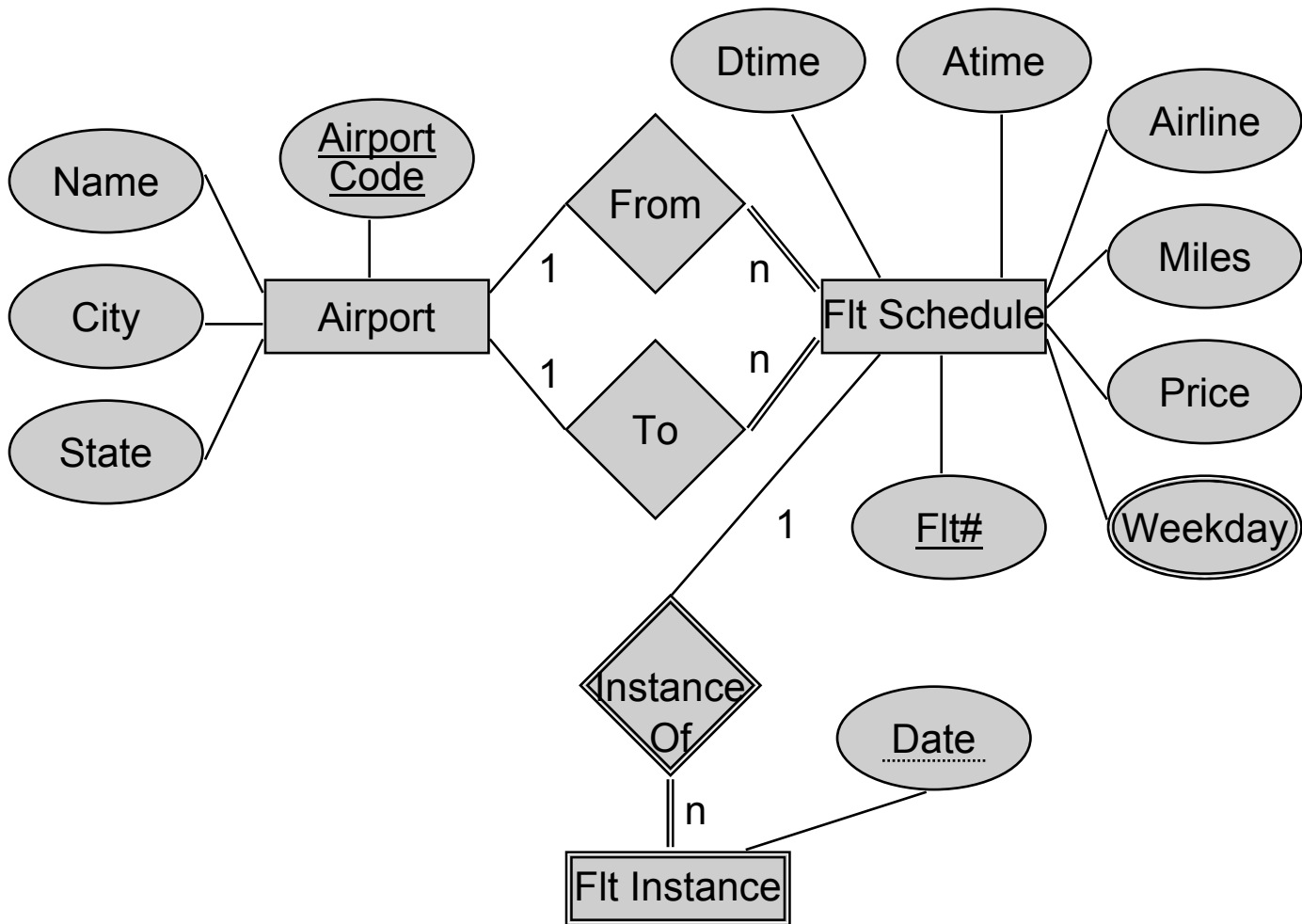


Example ER-Diagram

Create Flight Instance

Date: (yy-mm-dd)

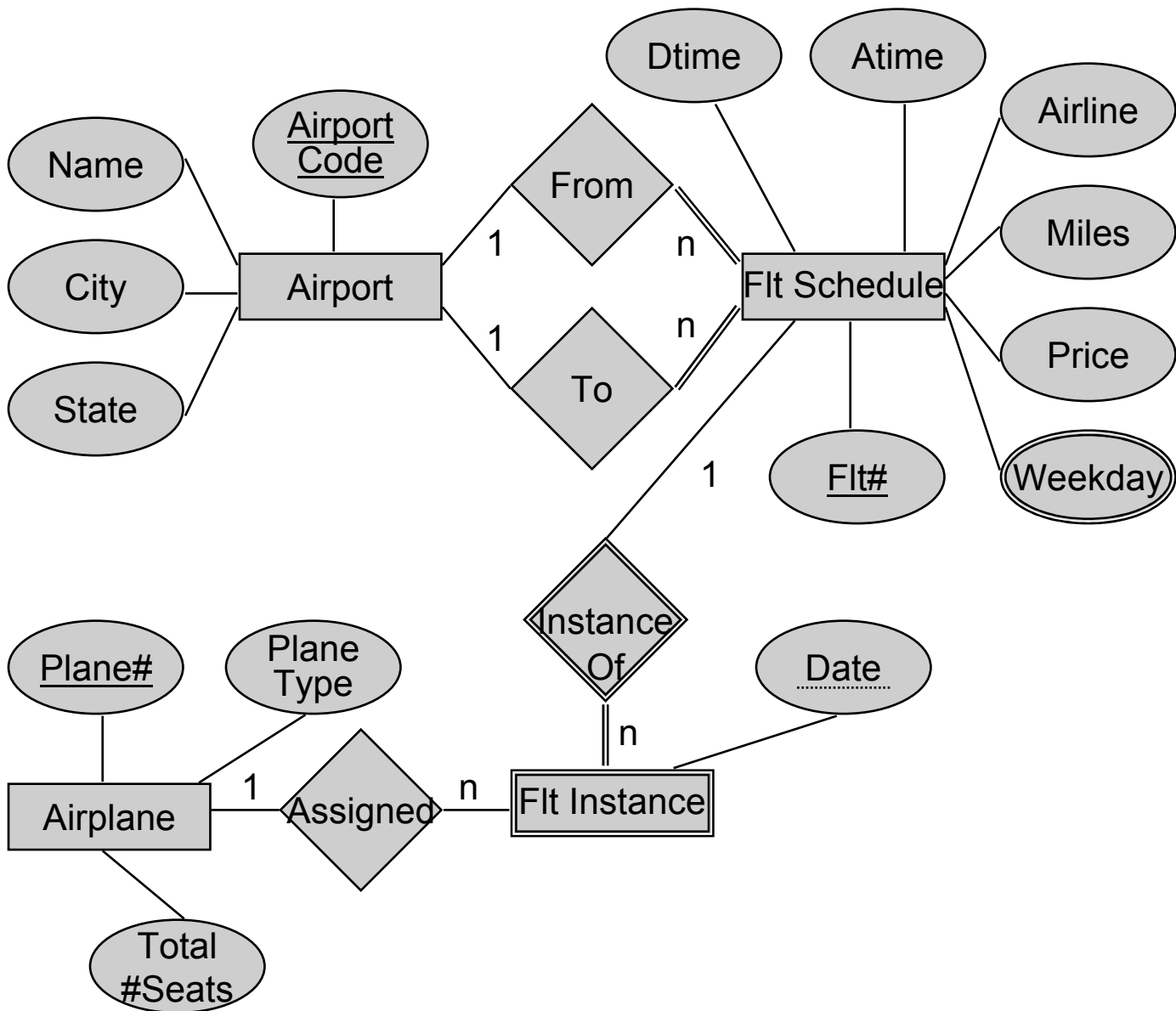
Flt#:



Example ER-Diagram

Airplanes		
Plane#	Plane Type	Total #Seats
<input type="text"/>	<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>	<input type="text"/>
-	-	-
-	-	-

Assign Flight	
Date:	<input type="text"/> (yy-mm-dd)
Flt#:	<input type="text"/>
Plane#:	<input type="text"/>



Example ER-Diagram

Reservation

Flight No: Flight Date:

Customer Info

First Name:

Middle Name:

Last Name:

Street Addr:

City:

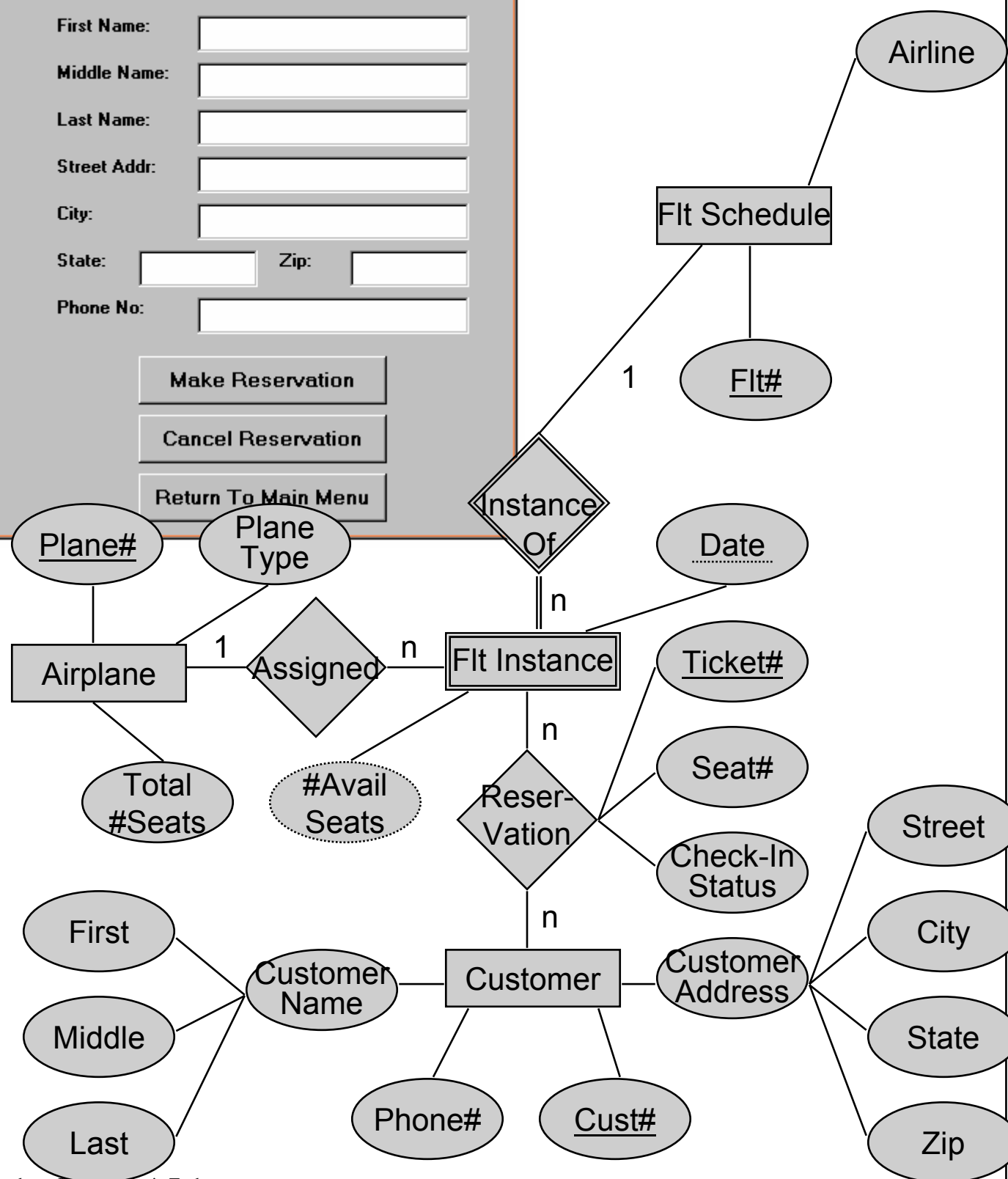
State: Zip:

Phone No:

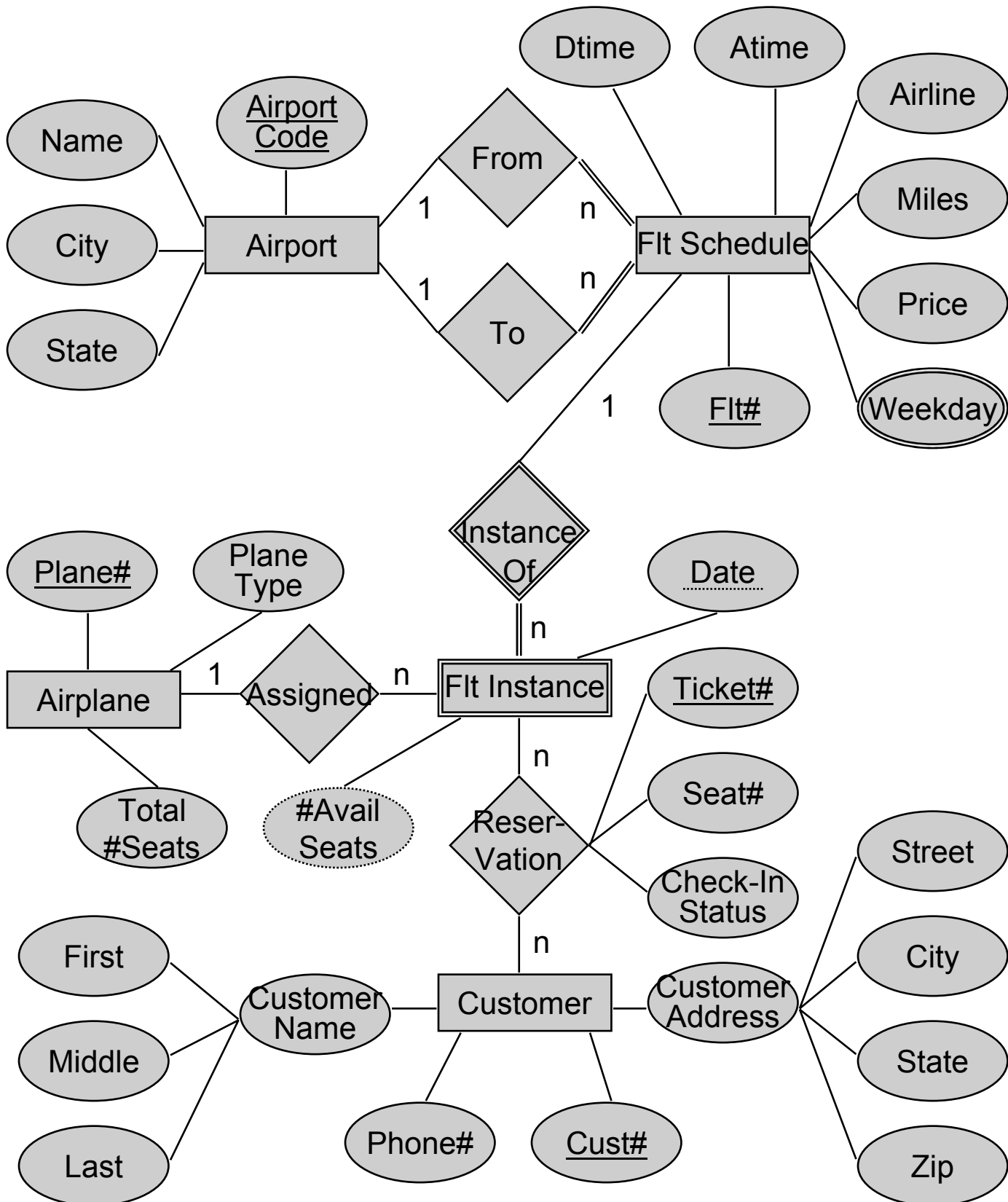
Make Reservation

Cancel Reservation

Return To Main Menu



Example ER-Diagram



Example Data Representation (from external documents)

- Flt-Schedule:
 - Flt#: LLDDD, like DL242, SK912, ...
 - Dtime, Atime: HH:MM:SS (time of day), like 09:30:00, 16:25:00, ...
(time zones? flights crossing midnight?)
 - Airline: L...L (30), like Delta, Scandinavian,
 - Miles: DDDD, like 500, 2550, ...
 - Price: DDDD.DD (US\$), like 725.00
 - Weekday: {MO,TU,WE,TH,FR,SA,SU}
- Airport:
 - Airport-Code: LLL, like ATL, CPH, ...
 - Name: L...L (30), like Hartsfield, Kastrup, ..
 - City: L...L (30), like Atlanta, København, ...
 - State: LL, like GA, MD, ...
(international addresses?)
- Flt-Instance:
 - Date: YYYY-MM-DD, like 1999-01-31
- etc.

Example Constraints

- **...must depart before arriving...**
 $\forall x \in \text{Flt-Schedule}: x.\text{Dtime} < x.\text{Atime}$
- **..cannot depart and arrive at same airport..**
 $\forall x \in \text{Flt-Schedule}: x.\text{From.Airport} \neq x.\text{To.Airport}$
- **...plane can only be in one place at a time..**
 $\forall x, y \in \text{Flt-Instance}, x \neq y, x.\text{Date} = y.\text{Date}, x.\text{Assigned.Airplane} = y.\text{Assigned.Airplane}:$
 $x.\text{Instance-Of.Flt-Schedule.Atime} <$
 $y.\text{Instance-Of.Flt-Schedule.Dtime}$ **or**
 $x.\text{Instance-Of.Flt-Schedule.Dtime} >$
 $y.\text{Instance-Of.Flt-Schedule.Atime}$
- **...match flight date and weekday...**
 $\forall x \in \text{Flt-Instance}: \text{Convert}(x.\text{Date to Weekday}) \in x.\text{Instance-of.Flt-Schedule.Weekday}$
- **...overbook by less than 10%...**
 $\forall x \in \text{Flt-Instance}: x.\#\text{Avail-Seats} = x.\text{Assigned.Airplane.Total\#Seats} \times 1.1 - \text{count}(x.\text{Reservation})$
- **..flights crossing midnight....time zones..**
- **many, many more**

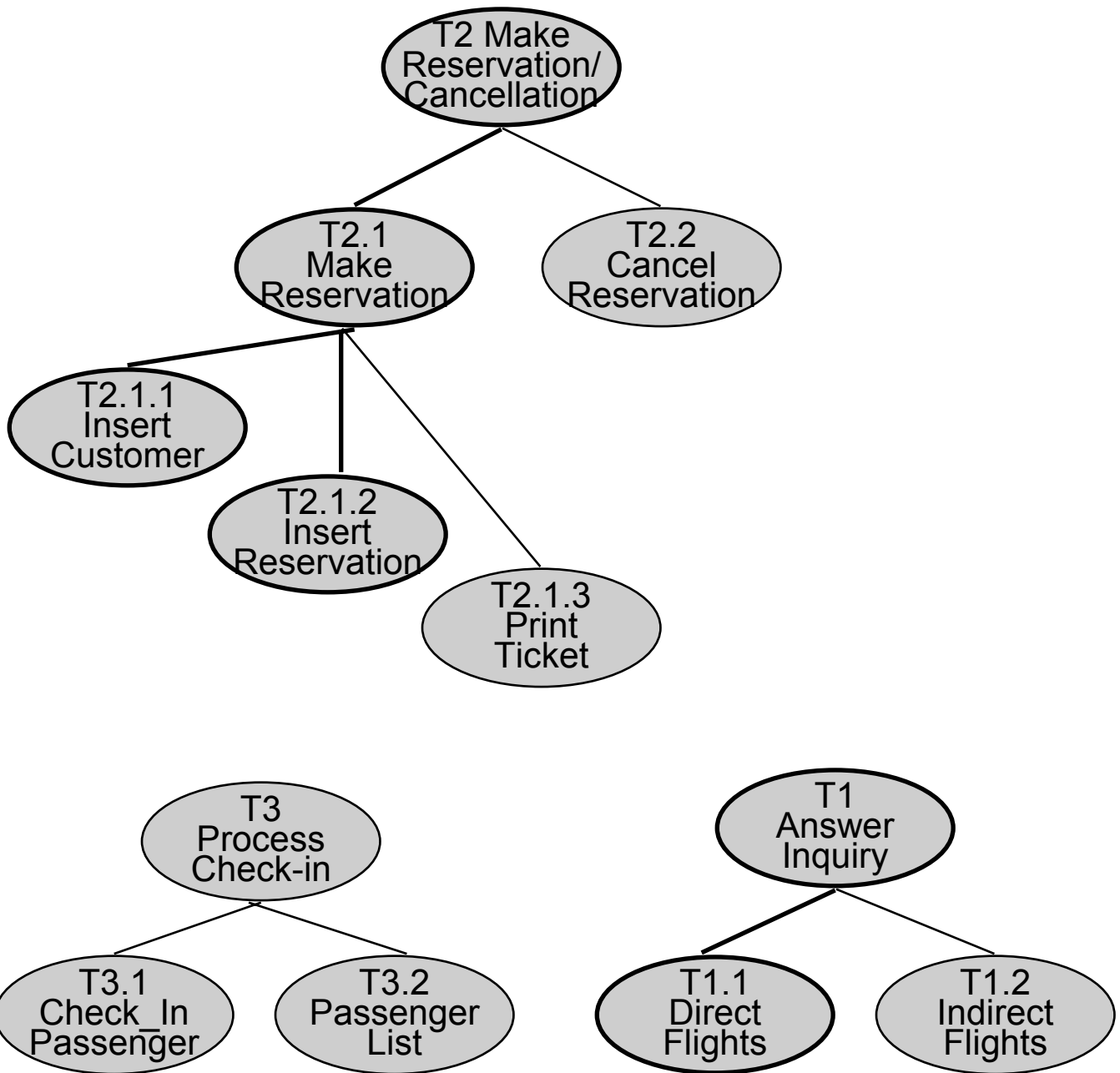
Task Forms

Task Name:	Unique name
Task Number:	Unique number, e.g. 1, 2, 3, ... Dot-notation for subtasks, e.g. 1.1, 1.2, ...
Description:	Brief natural language description of task
Enabling Cond.:	Description of what enables the task, e.g. information, control, time, ...
Frequency:	Frequency of task; use same uom across tasks, e.g. #times/day
Input:	List of fields from external input documents; List of entities and relationships from ER-Diagram
Output:	List of fields from external output documents; List of entities and relationships from ER-Diagram
Operation:	Detailed pseudo-code description of the task wrt. the external documents and the ER-Diagram
Subtasks:	List of subtasks controlled by the task.

Task Decomposition - rules of thumb

- Different enabling conditions apply to different parts of the task
 - may hold back parts of task able to run
- Different frequencies apply to different parts of the task
 - results in unnecessary costly indexing
- Different parts of ER-Diagram used by different parts of the task
 - may lock too large parts of database causing lock contention
- Many subtasks controlled by the task
 - may lock database too long causing lock contention
- Many diversified operations carried out by the task
 - difficult to understand and program

Example Task Decomposition



Example Task Statistics

Answer Inquiry (T1) = 360,000/day

3 airlines x 100 planes x 4 flights/plane/day x 250 seats/plane
x 1.1 seats booked + 30,000 additional inquiries

Direct-Flights (T1.1) = 360,000/day

Indirect-Flights (T1.2) = 39,600/day

10% of 360,000/day 2-leg + 1% of 360,000/day n-leg

Make-Reservation-Cancellation (T2): See subtasks.

Make-Reservation (T2.1) = 330,000/day

Insert-Customer (T2.1.1) = 16,500/day

5% of 330,000/day

Insert-Reservation (T2.1.2) = 330,000/day

Print-Ticket (T2.1.3) = 330,000/day

Cancel-Reservation (T2.2) = 99,000/day

30% of 330,000/day

Process-Check-In (T3): See subtasks.

Check-In-Passenger (T3.1) = 231,000/day

330,000/day - 99,000/day

Passenger-List (T3.2) = 1200/day

3 airlines x 100 planes x 4 flights/plane/day

Example Task Form

Task Name:	Answer-Inquiry
Task Number:	T1
Description:	Takes an Inquiry as input. Returns direct (and 2-leg) flights (if More Options are requested).
Enabling Cond.:	Receipt of an Inquiry
Frequency:	360,000/day.
Input:	EDs: Inquiry E-Types: Airport; Flt-Schedule R-Types: From; To
Output:	Inquiry
Operation:	Make Inquiry form visible While ReturnToMain button not pressed If DirectFlight or MoreOptions button pressed then validate entered data convert Date to Weekday make Inquiry form invisible If DirectFlight button pressed then call Direct-Flights(From, To, Weekday) Else If MoreOptions button pressed then call Indirect-Flights(From, To, Weekday) EndWhile Make Inquiry form invisible Return to Main
Subtasks:	Direct-Flights; Indirect-Flights;

Example Task Form

Task Name: Direct-Flights

Task Number: T1.1

Description: Takes Departure Airport, Arrival Airport and Date.
Returns information about all direct flights, if any.

Enabling Cond.: Receipt of an Inquiry.
Called from Answer-Inquiry.

Frequency: 360,000/day

Input: EDs: Inquiry
E-Types: Airport; Flt-Schedule
R-Types: From; To

Output: InquiryResult

Operation: Make InquiryResult form visible
IF EXISTS Flt-Schedule entity, such that:
 From.Airport.Airport-Code=:From
 and To.Airport.Airport-Code=:To
 and Weekday=:Weekday
THEN WHILE more Flt-Schedule entities DO
 PRINT(InquiryResult,
 :Flt#=Flt#
 :From=From.Airport.Airport-Code
 :To=To.Airport.Airport-Code
 :Dtime=Dtime
 :Atime=Atime);
While ReturnToInquiryMenu not pressed Do
EndWhile
Make InquiryResult form invisible
Return to Answer-Inquiry

Example Task Form

Task Name:	Make-Reservation/Cancellation
Task Number:	T2
Description:	This task supports requests for and cancellations of reservations, and printing of tickets
Enabling Cond.:	Receipt of Make Reservation/Cancellation request
Frequency:	See subtasks
Input:	EDs: Reservation/Cancellation E-Types: Flt-Schedule, Flt-Instance, Customer R-Types: Instance-Of, Reservation
Output:	EDs: Reservation/Cancellation E-Types: Flt-Instance, Customer R-Types: Reservation
Operation:	Make Reservation form visible While no buttons' been pressed Do EndWhile If Make-Reservation button pressed then MakeReservation Else If Cancel-Reservation been pressed then CancelReservation Else If ReturnToMainMenu button pressed the Return to Main
Subtasks:	Make-Reservation; Cancel-Reservation;

Example Task Form

Task Name: Make-Reservation

Task Number: T2.1

Description: This task makes a reservation for a known flight and enters customer information, if needed

Enabling Cond.: Receipt of Reservation/Cancellation with Make-Reservation=true;
Called from Make-Reservation/Cancellation(T2)

Frequency: 330,000/day

Input: EDs: Reservation/Cancellation
E-Types: Flt-schedule; Flt-Instance; Customer
R-Types: Instance-Of; Reservation

Output: EDs: Ticket
E-Types: Flt-Instance; Customer
R-Types: Reservation

Operation: IF NOT EXISTS Flt-Instance, such that
Date=:Flight-Date and
Instance-Of.Flt#=:Flight-No and
#Avail-Seats>0 THEN PRINT("No such flight")
ELSE IF NOT EXISTS Customer, such that
Customer-Name=(:First,:Middle,:Last) and
Customer-Address=(:Street,:City,:State,:Zip)
and Phone#=:Phone# THEN Cust#=:Cust#
THEN Insert-Customer;
PRINT("Customer inserted")
Insert-Reservation;
Print-Ticket;
Return to Main

Subtasks: Insert-Customer; Insert-Reservation; Print-Ticket;

Example Task Form

Task Name: Insert-Customer

Task Number: T2.1.1

Description: Insert new customer name, phone# and address

Enabling Cond.: Available Customer information
Called from Make-Reservation (T2.1)

Frequency: 16,500/day

Input: EDs: None
E-Types: None
R-Types: None

Output: EDs: None
E-Types: Customer
R-Types: None

Operation: insert into Customer
Values (new(:Cust#), :First, :Middle, :Last,
:Phone#, :Street, :City, :State, :Zip);
return Cust#=:Cust#;

Subtasks: None

Example Task Form

Task Name: Insert-Reservation

Task Number: T2.1.2

Description: Inserts Reservation on known Flt-Instance
for existing Customer

Enabling Cond.: Available Customer and Flt-Instance information
Called from Make-Reservation (T2.1)

Frequency: 330,000/day

Input: EDs: None
E-Types: None
R-Types: None

Output: EDs: None
E-Types: None
R-Types: Reservation

Operation: insert into Reservation Values
(Flt-Instance (:Flt#, :Date), Customer (:Cust#),
Seat# NULL, CheckInStatus NO,
Ticket# new(:Ticket#));

Subtasks: None

Design

Design

Purpose:

- create detailed design of normalized relational database schema
- create detailed design of tasks using abstract code with embedded SQL
- identify need for views

Input:

- EDs, ER-Diagram, TFs

Output:

- relational schema w/primary and foreign keys, constraint definitions in SQL, abstract code w/SQL, view definitions

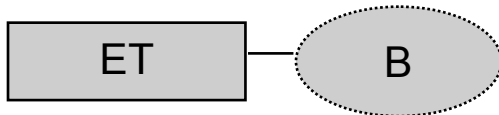
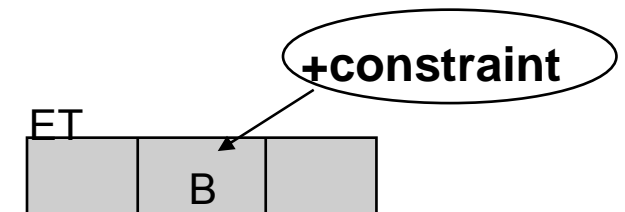
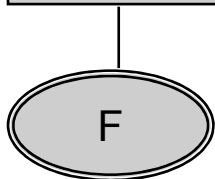
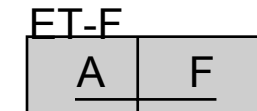
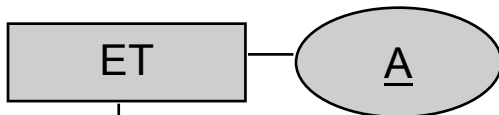
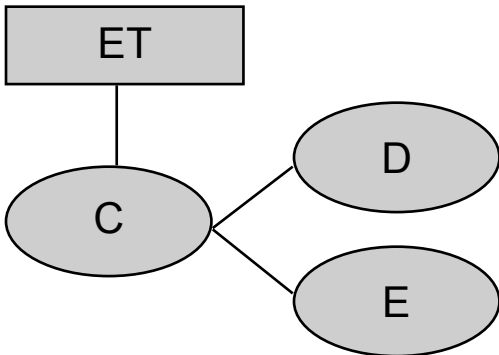
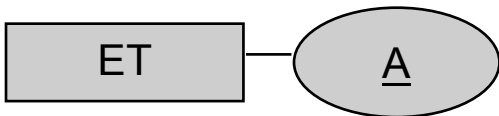
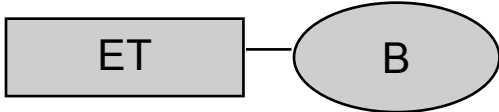
Techniques:

- database normalization; abstract coding

Tools:

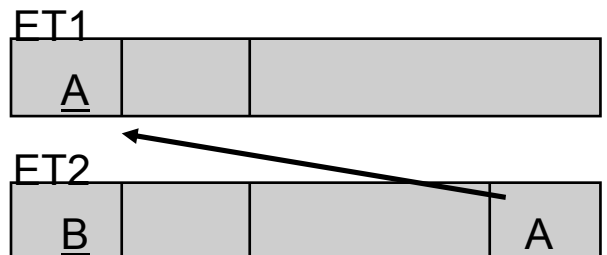
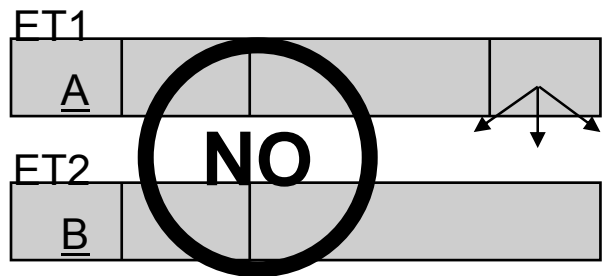
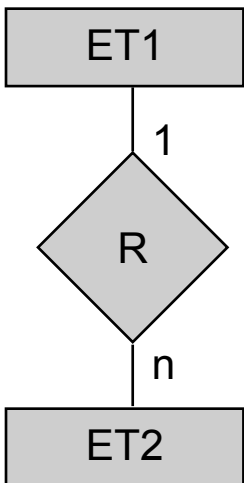
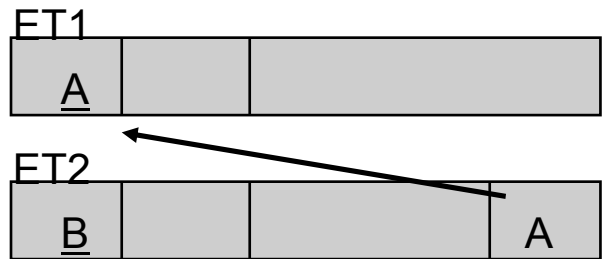
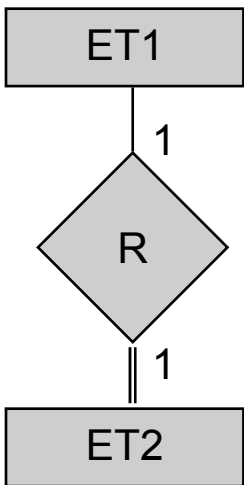
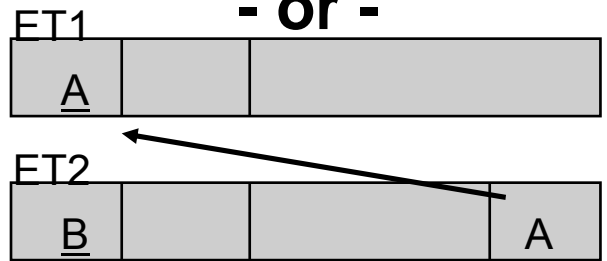
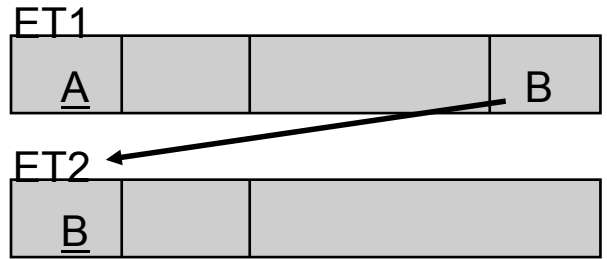
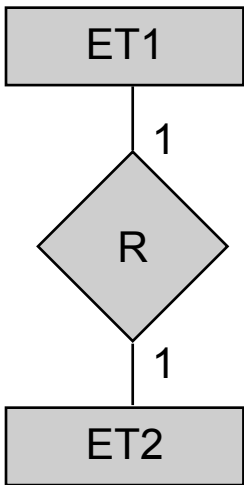
- mapping: ER-Model → Relational Model
- graphical DDLs
- abstract code; SQL; views

ER-Model → Relational Model

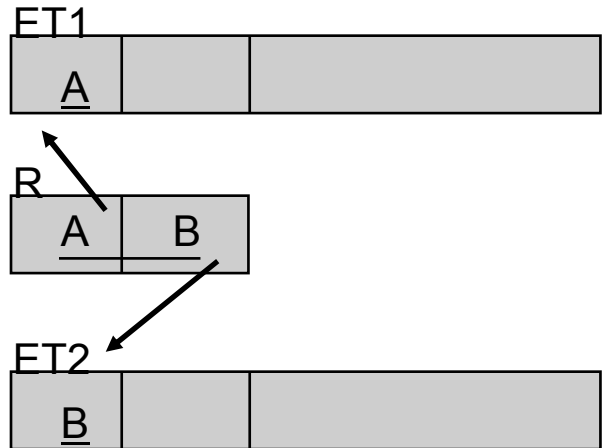
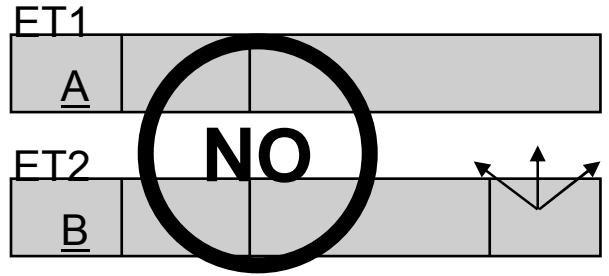
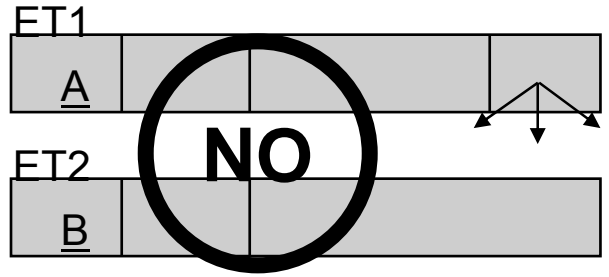
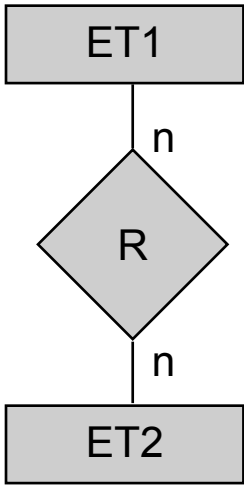


or,
define as a view

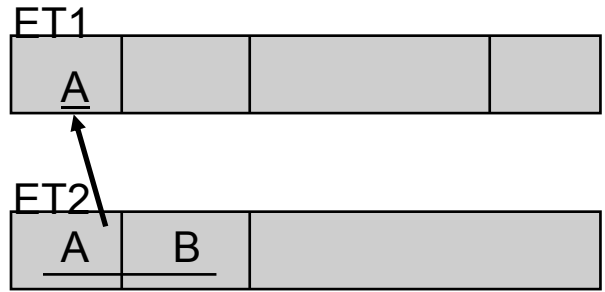
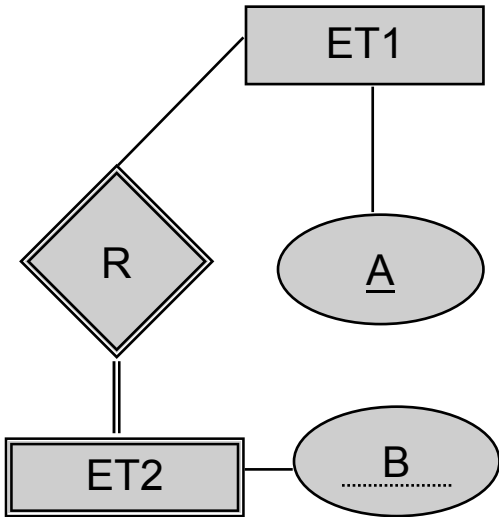
ER-Model → Relational Model



ER-Model → Relational Model



ER-Model → Relational Model



Example Relational Schema

AIRPORT

<u>airportcode</u>	name	city	state
--------------------	------	------	-------

FLT-SCHEDULE

<u>flt#</u>	airline	dtime	from-airportcode	atime	to-airportcode	miles	price
-------------	---------	-------	------------------	-------	----------------	-------	-------

FLT-WEEKDAY

<u>flt#</u>	<u>weekday</u>
-------------	----------------

FLT-INSTANCE

<u>flt#</u>	<u>date</u>	<u>plane#</u>	<u>#avail-seats</u>
-------------	-------------	---------------	---------------------

AIRPLANE

<u>plane#</u>	<u>plane-type</u>	<u>total-#seats</u>
---------------	-------------------	---------------------

CUSTOMER

<u>cust#</u>	first	middle	last	phone#	street	city	state	zip
--------------	-------	--------	------	--------	--------	------	-------	-----

RESERVATION

<u>flt#</u>	<u>date</u>	<u>cust#</u>	<u>seat#</u>	<u>check-in-status</u>	<u>ticket#</u>
-------------	-------------	--------------	--------------	------------------------	----------------

Example Relational Schema (primary and foreign keys)

AIRPORT

<u>airportcode</u>	name	city	state
--------------------	------	------	-------

FLT-SCHEDULE

<u>flt#</u>	airline	dtime	from-airportcode	atime	to-airportcode	miles	price
-------------	---------	-------	------------------	-------	----------------	-------	-------

FLT-WEEKDAY

<u>flt#</u>	<u>weekday</u>
-------------	----------------

FLT-INSTANCE

<u>flt#</u>	<u>date</u>	plane#	#avail-seats
-------------	-------------	--------	--------------

AIRPLANE

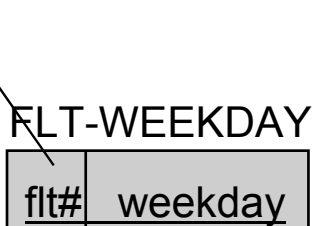
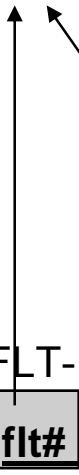
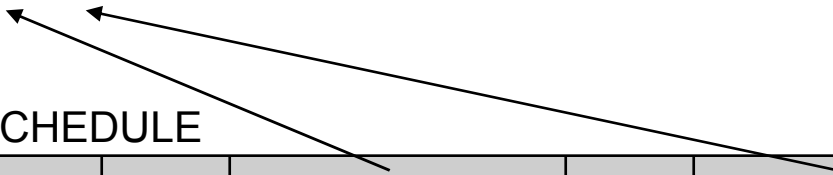
<u>plane#</u>	plane-type	total-#seats
---------------	------------	--------------

CUSTOMER

<u>cust#</u>	first	middle	last	phone#	street	city	state	zip
--------------	-------	--------	------	--------	--------	------	-------	-----

RESERVATION

<u>flt#</u>	date	<u>cust#</u>	seat#	check-in-status	<u>ticket#</u>
-------------	------	--------------	-------	-----------------	----------------



Database Normalization

1NF

- Are all the attribute values atomic?

2NF

- Do all attributes outside of the key functionally depend on the full key?

3NF

- Do any of the attributes outside of the key functionally depend on each other?

BCNF

- Are all determinants for functional dependencies candidate keys?

Database Normalization

The Good News:

- If you have designed the ER-Diagram well you don't need to 😊

The Bad News:

- Plane-type determines total-#seats in AIRPLANE ☹️
- (from-airportcode, to-airportcode) determine miles in FLT-SCHEDULE

The Ugly News:

- Someone else may have designed the ER-Diagram ☹️
- Database performance may not be acceptable ☹️

Example Relational Schema (constraints)

- **..must depart before arriving..**

```
CREATE ASSERTION IC-1 CHECK  
( NOT EXISTS (  
  SELECT * FROM FLT-SCHEDULE  
  WHERE DTIME ≥ ATIME));
```

- **..cannot depart and arrive at same airport..**

```
CREATE ASSERTION IC-2 CHECK  
( NOT EXISTS (  
  SELECT * FROM FLT-SCHEDULE  
  WHERE FROM-AIRPORTCODE=TO-AIRPORTCODE));
```

- **..plane can only be in one place at a time..**

```
CREATE ASSERTION IC-3 CHECK  
( NOT EXISTS (  
  SELECT X.*, Y.*  
  FROM (FLT-SCHEDULE NATURAL JOIN FLT-INSTANCE) X,  
  FROM (FLT-SCHEDULE NATURAL JOIN FLT-INSTANCE) Y  
  WHERE X.DATE=Y.DATE AND X.PLANE#≠Y.PLANE# AND  
  (X.DTIME, X.ATIME) OVERLAPS (Y.DTIME, Y.ATIME)));
```

- **..flights crossing midnight...time zones..**
- **..many, many more**

Example Abstract Code w/SQL

Direct-Flights T1.1

```
/* read(Inquiry, :Departure-Airport, :Arrival-Airport,:Date); */  
/* convert :Date to :Weekday; */
```

```
EXEC SQL WHENEVER NOT FOUND GOTO endloop;
```

```
EXEC SQL DECLARE DIRECT-FLIGHTS CURSOR FOR  
SELECT FROM-AIRPORTCODE, TO-AIRPORTCODE,  
       FLT-SCHEDULE.FLT#, DTIME, ATIME  
FROM FLT-SCHEDULE, FLT-WEEKDAY  
WHERE FLT-SCHEDULE.FLT#=FLT-WEEKDAY.FLT#  
AND FROM-AIRPORTCODE=:Departure-Airport  
AND TO-AIRPORTCODE=:Arrival-Airport AND WEEKDAY=:Weekday  
ORDER BY DTIME;
```

```
EXEC SQL OPEN DIRECT-FLIGHTS;
```

```
while
```

```
    EXEC SQL FETCH DIRECT-FLIGHTS  
    INTO :From, :To, :Flt#, :Dtime, :Atime;  
    write(Inquiry, :From, :To, :Flt#, :Date, :Dtime, :Atime)
```

```
endwhile;
```

```
endloop:
```

```
Exec SQL CLOSE DIRECT-FLIGHTS;
```

Example Abstract Code w/SQL

Make-Reservation T2.1

read(Reservation/Cancellation, :Flt#, :Date);

EXEC SQL WHENEVER SQLERROR GOTO QUIT;

EXEC SQL SELECT FLT#, DATE, #AVAIL-SEATS INTO :FL, :DA, :AV
FROM FLT-INSTANCE

WHERE FLT#=:Flt# AND DATE=:Date;

if NOT FOUND then

 write(Reservation/Cancellation, "No such flight")

else { if AV=0 then

 write(Reservation/Cancellation, "No available seats")

 else {

 read(Reservation/Cancellation, :First, :Middle,

 :Last, :Phone#, :Street, :City, :State, :Zip);

EXEC SQL SELECT CUST# INTO :Cust#

FROM CUSTOMER

WHERE FIRST=:First AND MIDDLE=:Middle AND LAST=:Last

AND STREET=:Street AND CITY=:City AND STATE=:State

AND ZIP=:Zip AND PHONE=:Phone;

if NOT FOUND then :Cust#=Insert-Customer

 (:First, :Middle, :Last, :Phone#, :Street, :City, :State,

:Zip);

 Insert-Reservation(:Flt#, :Date, :Cust#);

 Print-Ticket; }}

Quit:

if SQLERROR then EXEC SQL ROLLBACK WORK

else EXEC SQL COMMIT WORK;

Example Abstract Code w/SQL

```
Insert-Customer(:First,:Middle,:Last,:Phone#,:Street,:City,:State, :Zip);  
EXEC SQL INSERT INTO CUSTOMER  
VALUES( new(Cust#), :First, :Middle, :Last,  
       :Phone#, :Street, :City, :State, :Zip);  
return Cust#;
```

Implementation

Implementation

Purpose:

- create conceptual schema
- create internal schema
- implement abstract code

Input:

- relational schema w/primary and foreign keys, data representation, constraints in SQL, abstract code w/SQL, task decompositions, view definitions

Output:

- conceptual schema, internal schema, host-language code w/embedded SQL

Tools:

- SQL, host-language, LAPs
- relational database management system, pre-compiler
- host-language compiler

Example Conceptual Schema Implementation

```
CREATE DOMAIN AIRPORT-CODE CHAR(3)
CREATE DOMAIN FLIGHTNUMBER CHAR(5);
CREATE DOMAIN WEEKDAY CHAR(2)
      CONSTRAINT DAYS CHECK ( VALUE IN
        ('MO','TU','WE','TH','FR','SA','SU'));
```

```
CREATE TABLE FLT-SCHEDULE
(FLT#           FLIGHTNUMBER NOT NULL,
AIRLINE        VARCHAR(25),
DTIME          TIME,
FROM-AIRPORTCODE AIRPORT-CODE,
ATIME          TIME,
TO-AIRPORTCODE AIRPORT-CODE,
MILES          SMALLINT,
PRICE          DECIMAL(7,2),
PRIMARY KEY (FLT#),
FOREIGN KEY (FROM-AIRPORTCODE)
REFERENCES
      AIRPORT(AIRPORTCODE),
FOREIGN KEY (TO_AIRPORTCODE) REFERENCES
      AIRPORT(AIRPORTCODE));
```

Example Conceptual Schema Implementation

```
CREATE TABLE FLT-WEEKDAY  
(FLT#          FLIGHTNUMBER NOT NULL,  
WEEKDAY       WEEKDAY,  
UNIQUE(FLT#, WEEKDAY),  
FOREIGN KEY (FLT#) REFERENCES  
          FLT-SCHEDULE(FLT#));
```

```
CREATE TABLE FLT-INSTANCE  
(FLT#          FLIGHTNUMBER NOT NULL,  
DATE          DATE NOT NULL,  
PLANE#        INTEGER,  
PRIMARY KEY(FLT#, DATE),  
FOREIGN KEY FLT# REFERENCES  
          FLT-SCHEDULE(FLT#),  
FOREIGN KEY PLANE# REFERENCES  
          AIRPLANE(PLANE#));
```

Example Task Implementation

some C code

Direct-Flights T1.1

```
/* read(Inquiry, :Departure-Airport, :Arrival-Airport,:Date); */  
/* convert :Date to :Weekday; */
```

more C code

```
EXEC SQL WHENEVER NOT FOUND GOTO endloop;
```

more C code

```
EXEC SQL DECLARE DIRECT-FLIGHTS CURSOR FOR  
SELECT FROM-AIRPORTCODE, TO-AIRPORTCODE,  
       FLT-SCHEDULE.FLT#, DTIME, ATIME  
FROM FLT-SCHEDULE, FLT-WEEKDAY  
WHERE FLT-SCHEDULE.FLT#=FLT-WEEKDAY.FLT#  
AND FROM-AIRPORTCODE=:Departure-Airport  
AND TO-AIRPORTCODE=:Arrival-Airport AND WEEKDAY=:Weekday  
ORDER BY DTIME;
```

more C code

```
EXEC SQL OPEN DIRECT-FLIGHTS;  
while  
    EXEC SQL FETCH DIRECT-FLIGHTS  
    INTO :From, :To, :Flt#, :Dtime, :Atime;  
    write(Inquiry, :From, :To, :Flt#, :Date, :Dtime, :Atime)  
endwhile;
```

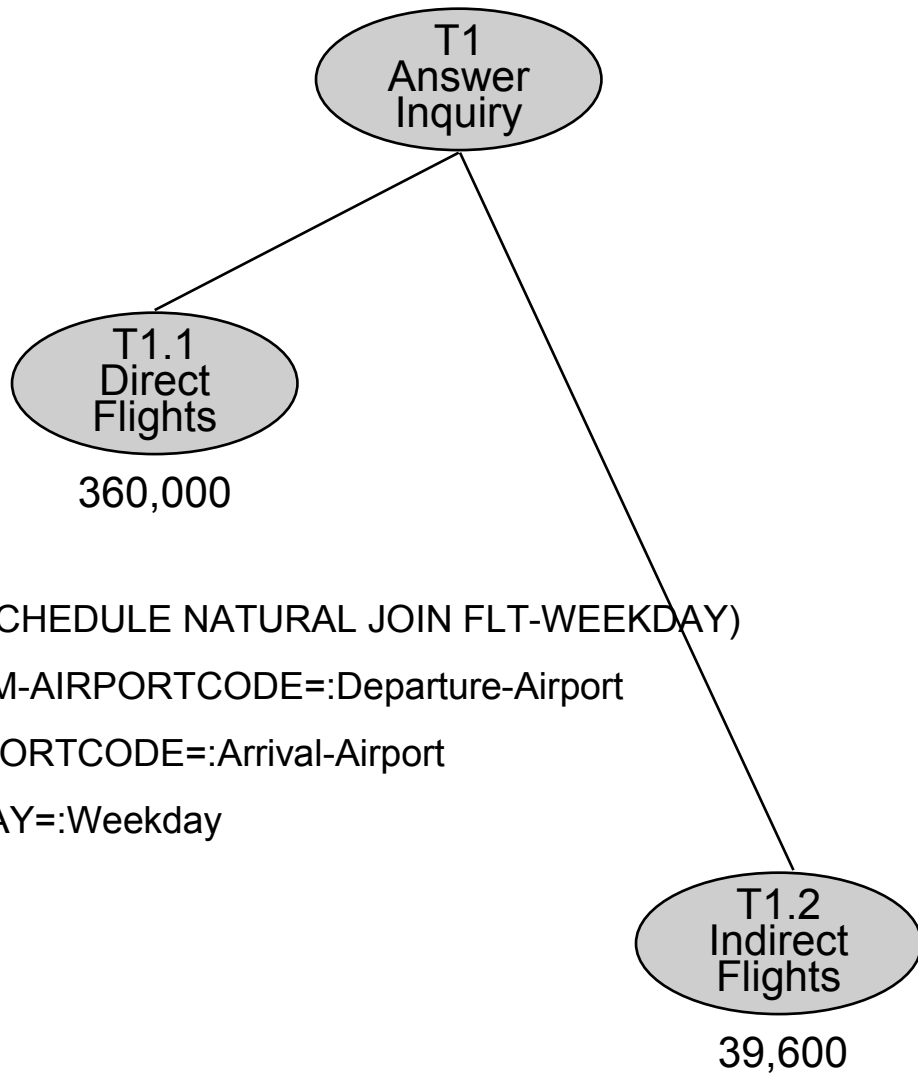
more C code

```
endloop:  
Exec SQL CLOSE DIRECT-FLIGHTS;
```

Example Task Implementation - VisualBasic

See separate set of transparencies

Example Logical Access Path



SELECT *

FROM (FLT-SCHEDULE NATURAL JOIN FLT-WEEKDAY)

WHERE FROM-AIRPORTCODE=:Departure-Airport

AND TO-AIRPORTCODE=:Arrival-Airport

AND WEEKDAY=:Weekday

SELECT *

FROM (FLT-SCHEDULE NATURAL JOIN FLT-WEEKDAY) X,

(FLT-SCHEDULE NATURAL JOIN FLT-WEEKDAY) Y

WHERE X.TO-AIRPORTCODE=Y.FROM-AIRPORTCODE

AND X.WEEKDAY=:WEEKDAY

AND X.WEEKDAY=Y.WEEKDAY

Example Logical Access Path

T2 Make
Reservation/
Cancellation

T2.1
Make
Reservation

T2.2
Cancel
Reservation

SELECT *
FROM FLT-INSTANCE
WHERE FLT#=... AND DATE=...

330,000

SELECT *
FROM CUSTOMER
WHERE CUSTOMER-NAME=...
AND CUSTOMER-ADDRESS=...
AND PHONE=...

99,000
DELETE
FROM RESERVATION
WHERE FLT#=...
AND DATE=...
AND NAME=...

T2.1.1
Insert
Customer

16,500

INSERT INTO CUSTOMER
VALUES

T2.1.2
Insert
Reservation

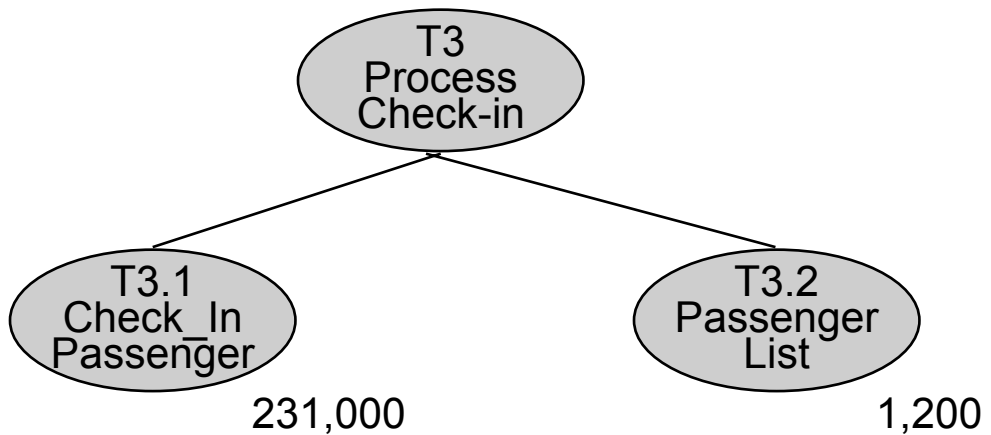
330,000

INSERT INTO RESERVATION
VALUES

T2.1.3
Print
Ticket

330,000

Example Logical Access Path



UPDATE RESERVATION
SET SEAT#=
WHERE FLT#=
AND DATE=
AND CUSTOMER-NAME=...

SELECT *
FROM RESERVATION
WHERE FLT#=
AND DATE=...

Example Relation Statistics

AIRPORT:

- record size: $3+30+30+2=65$ bytes
- # tuples: 42 tuples (6 hubs + 6 hubs * 6 non-hubs)
- # blocks: 1

FLT-SCHEDULE:

- record size: $5+30+6+3+6+3+4+8=65$ bytes
- # tuples: 2400 tuples assuming different workday and weekend schedules ($2 * 1200$)
- # blocks: 39

FLT-WEEKDAY:

- record size: $5+2=7$ bytes
- # tuples: 8400 tuples ($5 * 1200 + 2 * 1200$)
- # blocks: 15

FLT-INSTANCE:

- record size: $5+8+4+4=21$
- # tuples: 108,000 tuples (6 month flight schedule with half of the flights instantiated)
- # blocks: 554

Example Relation Statistics

AIRPLANE:

- record size: $4+1+4=9$ bytes
- # tuples: 300 tuple
- # blocks: 1

CUSTOMER:

- record size: $4+15+15+30+8+30+20+2+4=128$
- # tuples: 9,405,000 tuples (330,000 reservations per day, 95% by existing customers flying 1 time per month; $330,000 * .95 * 30$)
- # blocks: 294,000

RESERVATIONS:

- record size: $5+8+4+4+1+4=25$ bytes
- # tuples: 3,465,000 tuples (at any given time, about half of the reservations for the customers who will travel the following 30 days are in the database; $231,000 * 30 * .5$)
- # blocks: 21,150

Internal Schema Implementation

- **Primary file organization and indices** (clustering) are chosen to support the operations with the highest frequencies on the base relation
- **Secondary indices** (non-clustering) are introduced on a base relation if:
 - there is a relatively high probability for queries on the base relation
 - the queries are not supported by the primary file organization and indices
 - there is a relatively low probability for updates of the base relation

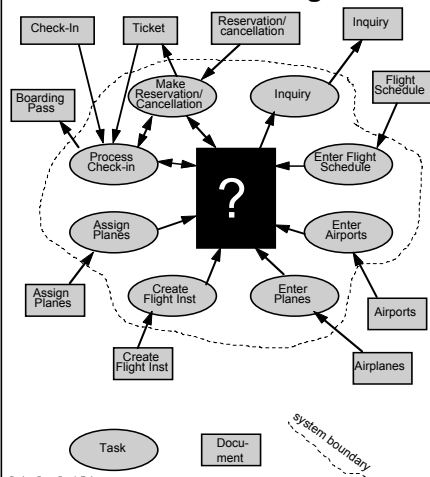
What Have We Learned?

External Documents

Inquiry Date: <input type="text"/> (yy-mm-dd) Departure Airport: <input type="text"/> Arrival Airport: <input type="text"/> More Options?: <input type="checkbox"/> (yes/no)		Create Flight Instance Date: <input type="text"/> (yy-mm-dd) Flt#: <input type="text"/>	
One-leg flights are: From To Flt# Date Dtime Atime		Assign Flight Date: <input type="text"/> (yy-mm-dd) Flt#: <input type="text"/> Plane#: <input type="text"/>	
Two-leg flights are:		Check-In/Seat selection Seat: <input type="text"/>	
Reservation/Cancellation Make Reservation <input type="checkbox"/> Cancel Reservation <input type="checkbox"/> Date: <input type="text"/> (yy-mm-dd) Flt#: <input type="text"/> Airline: <input type="text"/> Customer Name: <input type="text"/> Customer Address: <input type="text"/> First: <input type="text"/> Street: <input type="text"/> Middle: <input type="text"/> City: <input type="text"/> Last: <input type="text"/> State, Zip: <input type="text"/> Phone#: <input type="text"/>			

Database Group, Georgia Tech
© Leo Mark

Information Flow Diagram



Database Group, Georgia Tech
© Leo Mark

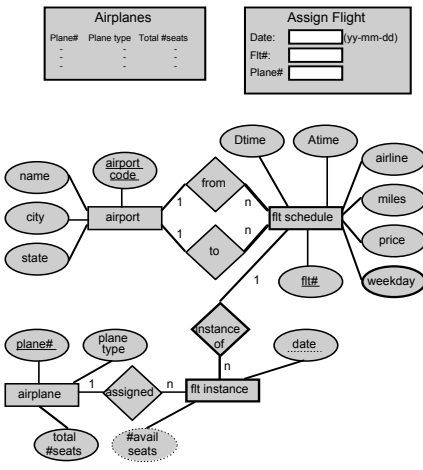


What goes into your database?

- 1 Everything in the database must come from somewhere
- 2 Everything on the input documents must go somewhere
- 3 Everything in the database must be used for something
- 4 Everything on the output documents must come from somewhere

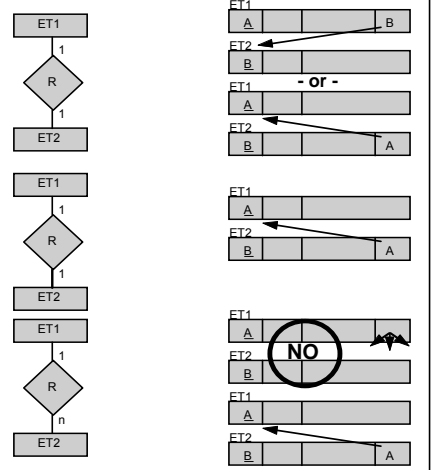
Database Group, Georgia Tech
© Leo Mark

ER-Diagram



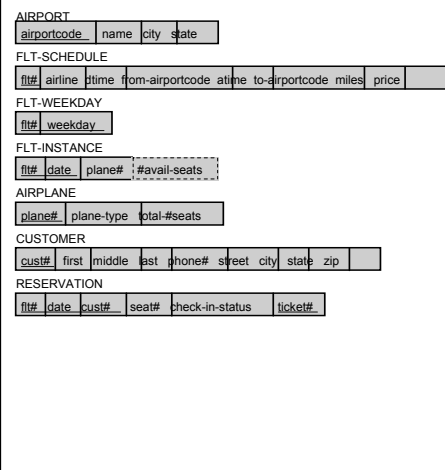
Database Group, Georgia Tech
© Leo Mark

ER-Model → Relational Model



Database Group, Georgia Tech
© Leo Mark

Relational Schema



Database Group, Georgia Tech
© Leo Mark

Example Conceptual Schema Implementation

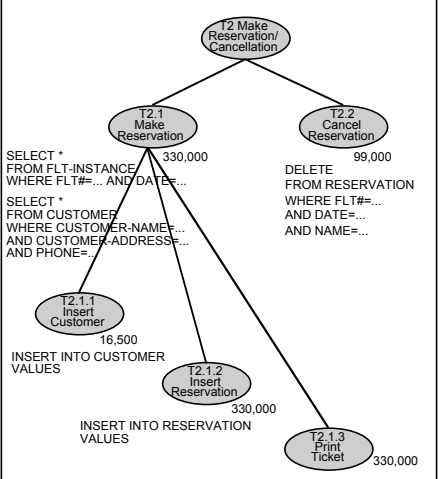
```

CREATE DOMAIN AIRPORT-CODE CHAR(3)
CREATE DOMAIN FLIGHTNUMBER CHAR(5);
CREATE DOMAIN WEEKDAY CHAR(2)
CONSTRAINT DAYS CHECK (VALUE IN ('MO','TU','WE','TH','FR','SA','SU'));

CREATE TABLE FLT-SCHEDULE
(FLT#          FLIGHTNUMBER NOT NULL ,
AIRLINE       VARCHAR(25),
DTIME         TIME,
FROM-AIRPORTCODE AIRPORT-CODE,
ATIME         TIME,
TO-AIRPORTCODE AIRPORT-CODE,
MILES         SMALLINT,
PRICE         DECIMAL(7,2),
PRIMARY KEY (FLT#),
FOREIGN KEY (FROM-AIRPORTCODE) REFERENCES AIRPORT(AIRPORTCODE),
FOREIGN KEY (TO-AIRPORTCODE) REFERENCES AIRPORT(AIRPORTCODE));
    
```

Database Group, Georgia Tech
© Leo Mark

Example Logical Access Path



Database Group, Georgia Tech
© Leo Mark

What Have We Learned?

This summary diagram shows the flow of information from external documents through various modeling stages to a logical access path. It includes a question mark icon and the same four principles listed in the top-right section:

- 1 Everything in the database must come from somewhere
- 2 Everything on the input documents must go somewhere
- 3 Everything in the database must be used for something
- 4 Everything on the output documents must come from somewhere

Database Group, Georgia Tech
© Leo Mark