Classical Database Development Methodology

Note that these slides are verbatim from the Addison-Wesley Web Site for the textbook and are provided for information purposes to demonstrate the entire design and development process from a database perspective. Note that data flow diagrams are used instead of UML diagrams.

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

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

<table>
<thead>
<tr>
<th>AIRLINE</th>
<th>From City</th>
<th>To City</th>
<th>Flt#</th>
<th>Dtime</th>
<th>Atime</th>
<th>Weekdays</th>
<th>Miles</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Boarding Pass

<table>
<thead>
<tr>
<th>Airline</th>
<th>Customer Name</th>
<th>From</th>
<th>To</th>
<th>Flt#</th>
<th>Date</th>
<th>Dtime</th>
<th>Atime</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Ticket

<table>
<thead>
<tr>
<th>Airline</th>
<th>Ticket#</th>
<th>Customer Name</th>
<th>From</th>
<th>To</th>
<th>Flt#</th>
<th>Date</th>
<th>Dtime</th>
<th>Atime</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Passenger List

<table>
<thead>
<tr>
<th>Date</th>
<th>Flt#</th>
<th>Airline</th>
<th>Customer Name</th>
<th>Seat#</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Airports

<table>
<thead>
<tr>
<th>Airport Code</th>
<th>Name</th>
<th>City</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Airplanes

<table>
<thead>
<tr>
<th>Plane#</th>
<th>Plane Type</th>
<th>Total Seats</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Create Flight Instance

<table>
<thead>
<tr>
<th>Date: (yy-mm-dd)</th>
<th>Flt#:</th>
</tr>
</thead>
</table>

#### Assign Flight

<table>
<thead>
<tr>
<th>Date: (yy-mm-dd)</th>
<th>Flt#:</th>
<th>Plane#</th>
</tr>
</thead>
</table>

#### Check-In/Seat selection

<table>
<thead>
<tr>
<th>Ticket#</th>
<th>Seat</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Inquiry

<table>
<thead>
<tr>
<th>Date: (yy-mm-dd)</th>
<th>Departure Airport:</th>
<th>Arrival Airport:</th>
<th>More Options? (yes/no)</th>
<th>One-leg flights are: From</th>
<th>To</th>
<th>Flt#</th>
<th>Date</th>
<th>Dtime</th>
<th>Atime</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Date: (yy-mm-dd)</th>
<th>Make Reservation</th>
<th>Cancel Reservation</th>
<th>Flt#:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Reservation/Cancellation

<table>
<thead>
<tr>
<th>Date: (yy-mm-dd)</th>
<th>Flt#:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Customer Name</th>
<th>Customer Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>First:</td>
<td>Street:</td>
</tr>
<tr>
<td>Middle:</td>
<td>City:</td>
</tr>
<tr>
<td>Last:</td>
<td>State, Zip:</td>
</tr>
<tr>
<td>Phone#:</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ticket#</th>
<th>Seat</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 flights, 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

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**

[Diagram showing the relationships between Airport, Name, City, and State]
Example ER-Diagram

Example ER-Diagram (integrate)
Example ER-Diagram

Create Flight Instance
Date: (yy-mm-dd)
Flt:

From

To

Atime

Dtime

Flt

Airline

Miles

Price

Weekday

Instance Of

Flight Schedule

Flt Instance

Example ER-Diagram

Airplanes

Plané# Plane Type Total #Seats
- - -

Assign Flight
Date: (yy-mm-dd)
Flt:

Plane#

Assign Flight

Flt Instance

Assign Flight

From

To

Atime

Dtime

Flt

Airline

Miles

Price

Weekday

Instance Of

Flight Schedule

Flt Instance

Airline

Miles

Price

Weekday

Total #Seats
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, ...
  - 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, ...

- **Flt-Instance**:
  - Date: YYYY-MM-DD, like 1999-01-31

...etc.

Example Constraints

- **...must depart before arriving...**
  \[ \forall x \in \text{Flt-Schedule}: x.Dtime < x.Atime \]

- **...cannot depart and arrive at same airport..**
  \[ \forall x \in \text{Flt-Schedule}: x.From.Airport \neq x.To.Airport \]

- **...plane can only be in one place at a time..**
  \[ \forall x,y \in \text{Flt-Instance}, x \neq y, x.Date=y.Date, x.Assigned.Airplane=y.Assigned.Airplane: x.\text{Instance-Of.Flt-Schedule}.Atime < y.\text{Instance-Of.Flt-Schedule}.Dtime \text{ or } x.\text{Instance-Of.Flt-Schedule}.Dtime > y.\text{Instance-Of.Flt-Schedule}.Atime \]

- **...match flight date and weekday...**
  \[ \forall x \in \text{Flt-Instance}: \text{Convert}(x.Date \text{ to Weekday}) \in x.\text{Instance-of.Flt-Schedule}.Weekday \]

- **...overbook by less than 10%...**
  \[ \forall x \in \text{Flt-Instance}: x.\#\text{Avail-Seats} = x.Assigned.Airplane.\text{Total}\#\text{Seats} \times 1.1 - \text{count}(x.\text{Reservation}) \]

- **...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, 2-leg, 3-leg, ... flights as long as 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:
        PRINT(Inquiry, "One-leg flights are:");
        Direct Flights;
        PRINT(Inquiry, "More Options?");
        READ(Inquiry, More Options);
        i=2;
        WHILE More Options DO
            PRINT(Inquiry, "The", i, "-leg flights are:");
            Indirect Flights(i);
            PRINT(Inquiry, "More Options?");
            READ(Inquiry, More Options);
            i+=1
        ENDWHILE;
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: Inquiry
Operation:
        READ(Inquiry, 
            :Departure-Airport, :Arrival-Airport,:Date);
        Convert :Date to :Weekday;
        IF EXISTS Flt-Schedule entity, such that: 
            From.Airport.Airport-Code=:Departure-Airport 
            and To.Airport.Airport-Code=:Arrival-Airport 
            and Weekday=:Weekday
        THEN WHILE more Flt-Schedule entities DO
            PRINT(Inquiry,
                :From=From.Airport.Airport-Code
                :To=To.Airport.Airport-Code
                :Flt#=Flt#
                :Dtime=Dtime
                :Atime=Atime);
### 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:**  
- IF Make-Reservation THEN Make-Reservation  
- ELSE  
  - IF Cancel Reservation THEN Cancel-Reservation;  
**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:**  
- READ(Reservation/Cancellation, :Flt#, :Date);  
- IF NOT EXISTS Flt-Instance entity, such that  
  - Date=:Date and Instance-Of.Flt#:Flt#  
  - and Avail-Seats>0 THEN STOP;  
- READ(Reservation/Cancellation, :First, :Middle, :Last, :Phone#, :Street, :City, :State, :Zip);  
- IF EXISTS Customer entity, such that  
  - Customer-Name=(:First,:Middle,:Last)  
  - and Customer-Address=(:Street,:City,:State,:Zip)  
  - and Phone#:Phone# THEN Cust#:Cust#;  
- ELSE Insert-Customer;  
- Insert-Reservation;  
- Print-Ticket;  
**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
(Flt-Instance (:Flt#, :Date), Customer (:Cust#),
Seat# NULL, CheckInStatus NO,
Ticket# new(:Ticket#));
```
**Subtasks:** None
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 -

ET-Model → Relational Model

- or -
ER-Model ➞ Relational Model

ET1

ET2

ET1

ET2

ER-Model ➞ Relational Model

ET1

ET2
Example Relational Schema

AIRPORT
airportcode name city state

FLT-SCHEDULE
flt airline dtime from-airportcode atime to-airportcode miles price

FLT-WEEKDAY
flt weekday

FLT-INSTANCE
flt date plane# #avail-seats

AIRPLANE
plane# plane-type total-#seats

CUSTOMER
cust# first middle last phone# street city state zip

RESERVATION
flt date cust# seat# check-in-status ticket#

Example Relational Schema (primary and foreign keys)

AIRPORT
airportcode name city state

FLT-SCHEDULE
flt airline dtime from-airportcode atime to-airportcode miles price

FLT-WEEKDAY
flt weekday

FLT-INSTANCE
flt date plane# #avail-seats

AIRPLANE
plane# plane-type total-#seats

CUSTOMER
cust# first middle last phone# street city state zip

RESERVATION
flt date cust# seat# check-in-status ticket#
### 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#;

Example Abstract Code w/SQL

Insert-Reservation(:Flt#, :Date, :Cust#);
EXEC SQL INSERT INTO RESERVATION
VALUES( new(Resv#), :Flt#, :Date, :Cust#);
return Resv#;
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));

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

```c
// 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 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=Y.WEEKDAY
```
**Example Logical Access Path**

**T2 Make Reservation/Cancellation**

- T2.1 Make Reservation
  - SELECT *
    - FROM FLT-INSTANCE
      - WHERE FLT#=... AND DATE=...
  - Insert Reservation
  - Print Ticket

- T2.2 Cancel Reservation
  - DELETE
    - FROM RESERVATION
      - WHERE FLT#=... AND DATE=... AND NAME=...
  - Insert Customer

**T2.1 Insert Customer**

- INSERT INTO CUSTOMER
  - VALUES

**T2.2 Insert Reservation**

- INSERT INTO RESERVATION
  - VALUES

**T2.3 Print Ticket**

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

**Example Logical Access Path**

**T3 Process Check-in**

- T3.1 Check In Passenger
  - UPDATE RESERVATION
    - SET SEAT=#
      - WHERE FLT#=... AND DATE=... AND CUSTOMER-NAME=...

- T3.2 Passenger List
  - 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: $5 + 30 + 6 + 3 + 6 + 4 + 8 = 65$ bytes
- # tuples: $2400$ tuples assuming different workday and weekend schedules ($2 \times 1200$)
- # blocks: $39$

**FLT-WEEKDAY:**
- record size: $5 + 2 = 7$ bytes
- # tuples: $8400$ tuples ($5 \times 1200 + 2 \times 1200$)
- # 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: $554$

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

**CUSTOMER:**
- record size: $4 + 15 + 30 + 8 + 30 + 20 + 2 + 4 = 128$
- # tuples: $9,405,000$ tuples ($330,000$ reservations per day, $95\%$ by existing customers flying $1$ time per month; $330,000 \times .95 \times 30$)
- # 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; $231,000 \times 30 \times .5$)
- # 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

Example Internal Schema Implementation

**FLT-SCHEDULE**: **FLT-WEEKDAY**:
- joined 360,000/day in Direct-Flights
- almost never updated
- naive join cost: 39+15=585 blocks
- very small relations; will easily fit in memory
- join cost without indices 39+15=54 blocks
- join cost with B+-tree primary indices on flt#: 39+15=54 blocks
- join cost with B+-tree primary index on from-airportcode:
  - 39*(18+5+9+6)=22400+15=2405=20 blocks
- using to-airportcode to reduce the 5 blocks found via from-airportcode as much as possible, i.e. to 5+18/288=1 block will not help since the 5 blocks are already in memory and the 1 block references 18 tuples randomly on 15 blocks of FLT-WEEKDAY
- the join cost with a B+-tree primary index on flt# in FLT-WEEKDAY will not be reduced because the 1 block of FLT-SCHEDULE still references 18 tuples on 15 blocks in FLT-WEEKDAY
- a B+-tree primary index on weekday will reduce FLT-WEEKDAY to 15/7=3 blocks
- total join cost with B+-tree primary index on from-airportcode and B+-tree primary index on weekday is 5+3=8 blocks
- a secondary index on to-airportcode will not speed up the join(s) needed for Indirect-Flights because the possible 41 to-airportcodes are randomly spread on 39 blocks
Example Internal Schema Implementation

FLT-INSTANCE:
- randomly accesses 330,000/day from Make-Reservation
- updated about 2.2% per day
- a primary hash index on the composite key (flt#, date) will guarantee an access cost of 1-2 blocks
- The hash index may have to be reorganized every two weeks. It will take approximately 6 seconds each time.

CUSTOMER:
- randomly accessed 330,000/day from Make-Reservation
- updated 16,500/day from Insert-Customer
- a primary hash index on the composite key (first, middle, last) will guarantee an access cost of 1-2 blocks and an insertion cost of 2-3 blocks
- insertions are relatively few; less than .18% per day or less than 16% in 3 months. If customers that have not flown for a year are purged every 3 months (a date-of-last-flight may be needed), the hash index will be relatively stable and could probably be filled more than 50%. Purging will take approximately 50 minutes each time.

RESERVATIONS:
- 330,000 insertions/day from Make-Reservation
- 99,000 deletions/day from Cancel-Reservation
- 231,000 deletions/day from Check-In
- 19% change/day. This is a very unstable relation.
- since all access is random a primary hash index on the composite key (flt#, date, cust#) would guarantee an update cost of 2-3 blocks
- the hash index should be filled no more than 50% and reorganization is required every day. Reorganization will take approximately 4 minutes each time.

Total processing time:

Direct-Flights: 360,000 * 8 * .01 sec = 8.00 hrs
Make-Reservation:
  check flt-instance: 330,000 * 2 * .01 sec = 1.83 hrs
  check customer: 330,000 * 2 * .01 sec = 1.83 hrs
Insert-Customer: 16,500 * 3 * .01 sec = 0.14 hrs
Insert-Reservation: 330,000 * 3 * .01 sec = 2.75 hrs
Cancel-Reservation: 99,000 * 3 * .01 sec = 0.83 hrs
Check-In: 231,000 * 3 * .01 sec = 1.93 hrs
TOTAL: 17.31 hrs