A Data Warehouse Solution for Analyzing RFID-Based Baggage Tracking Data

Tanvir Ahmed Torben Bach Pedersen Hua Lu
Department of Computer Science
Aalborg University
Aalborg, Denmark
Email: {tanvir, tbp, luhua}@cs.au.dk

Abstract—Today, airport baggage handling is far from perfect. Baggage goes on the wrong flights, is left behind, or gets lost, which costs a lot of money for the airlines, as well as frustration for the passengers. To remedy the situation, we present a data warehouse (DW) solution for storing and analyzing spatio-temporal Radio Frequency Identification (RFID) baggage tracking data. Analysis of this data can yield interesting results on baggage flow, the causes of baggage mishandling, and the parties responsible for the mishandling (airline, airport, handler,...), which can ultimately lead to improved baggage handling quality. The paper presents a carefully designed data warehouse (DW), with a relational schema sitting underneath a multidimensional data cube, that can handle the many complexities in the data. The paper also discusses the Extract-Transform-Load (ETL) flow that loads the data warehouse with the appropriate tracking data from the data sources. The presented concepts are generalizable to other types of multi-site indoor tracking systems based on Bluetooth and RFID. The system has been tested with large amount of real-world RFID-based baggage tracking data from a major industry initiative. The developed solution is shown to both reveal interesting insights as well as being several orders of magnitude faster than computing the results directly on the data sources.

Index Terms—RFID; data warehouse; data cube; data analysis; baggage tracking; moving objects; indoor tracking;

I. INTRODUCTION

A recent report\footnote{SITA Baggage Report 2012 www.sita.aero/content/baggage-report-2012} discloses the enormous loss caused by baggage mishandling in the aviation industry. Each year more than 31M passengers and 34M bags are affected by baggage mishandling which costs the aviation industry 3,300 M USD. A passenger wastes on average 1.7 days of his vacation or business trip waiting for the mishandled bag. Typical baggage mishandling problems include flight delay, bag loss, wrong bag destination, failure to load bags at the origin airport, missed connection at transit hubs, etc. During the travel from the origin airport to the final destination, a bag is moved over different places in multiple steps: check-in, screening, sortation, loading, transition, arrival, etc. In these steps, a bag is handed over between a large number of stakeholders. In contrast with the visibility of passengers’ movement steps during the end-to-end journey, the baggage movement is considerably less visible due to the very limited information to passengers. A single error or inefficacy in a handover can cause a bag not to reach its intended destination with its owner [12].

Radio Frequency Identification (RFID) technology is used in many applications for monitoring object movement. The use of RFID in baggage tracking systems enables to track a bag in an airport as well as along its travel route cross airports. RFID tracking systems generate huge amounts of data. For example, Walmart has recently started using tags at the item level and research firm Venture Development Corporation predicts that this will generate up to 7 terabytes of data per day [5]. In RFID-based airport baggage tracking systems, these huge amounts of RFID data can be very useful for analyzing and mining purposes. Coupled with other kinds of information about routes, transit airports, transit durations, flights and punctuality, airlines, handler, special events, etc., RFID baggage tracking data can reveal a lot of information about baggage handling quality. Analyzing these data will open the door to identifying the different problems in baggage handling and finding solutions for the problems. This is exactly the goal of the BagTrack project (www.daisy.aau.dk/bagtrack), within which this work took place. Here, a number of important industry players have teamed up with our data management team to revolutionize baggage handling using RFID.

In this paper, we present a multidimensional database warehouse solution for RFID-based baggage tracking data. To the best of our knowledge, this paper is the first to design a multidimensional data warehouse, including a relational DW schema with a data cube on top, for this important domain. The proposed data warehouse contributes to the airline baggage handling process by providing a framework for data analysis and answering complex queries that can ultimately improve the baggage handling quality. For example, the manager of Copenhagen airport may ask a query like how many bags were sent to wrong destination from Copenhagen airport in the Easter holidays of 2012. Another query can be find the average number of baggage traveling from Copenhagen airport in the afternoon of each Sunday. Our data warehouse supports such useful queries effectively and efficiently.

Our data warehouse design features several novelties. First, it captures not only the RFID baggage tracking data but also baggage flow along different dimensions like airport, airline, date and time, all at several levels of granularity. We also handle the complex many-to-many relationship between bag and flight effectively. Second, our design treats date and time as different dimensions, each with different hierarchical
levels. Each date is localized to a particular airport, to allow
capturing special local events, like strikes, sports games, etc.
This localization makes it easy to ask complex queries about
baggage movement in a particular event or occasion. Third,
our data warehouse design supports powerful data analysis
queries in a both easy and efficient way. The relevant results
enable the users (e.g., an airline) to find out the places (e.g., a
particular airport) where mishandling occurs most of the time,
and thus to determine the reasons of baggage mishandling. The
paper also discusses the complexities of performing Extract-
Transform-Load (ETL) to load the DW from the source data,
including our solutions to the encountered challenges.

The remainder of the paper is organized as follows. Section
2 reviews related work. Section 3 presents an overview of
RFID-based airport baggage handling process and the structure
of RFID data. Section 4 presents the relational data warehouse
design and the multidimensional data cube design. Section 5
represents the ETL steps to load the data from source schema
into the data warehouse. Section 6 presents the experimental
results. Finally, Section 7 concludes the paper and points to
the challenges for future work that we encountered.

II. RELATED WORK

Data warehousing, mining and work flow analysis tech-
niques have been proposed for RFID-based supply chain
systems [3], [4], [5], [6]. Those proposals take advantage of
bulky movement of objects in supply chains to compress the
massive RFID data. Efficient storage scheme and encoding
the object paths are designed [9], [10] to support faster query
performance for supply chain RFID data. RFID data warehous-
ing for tracking patients and drugs has been studied [2]. The
general challenges and solutions for RFID data management
are also discussed [1], [13].

The scenario of airport baggage management in this paper
differs from the settings of previous research. Unlike objects
in supply chain management, airport bags do not move in a
bulky way. In addition, the bag locations in airport baggage
management are much more fine grained so that places and
reasons of mishandling can be identified at a satisfactory
level. Further, the time dimension in airport baggage manage-
ment is also much more fine grained so that complex time-
dependent queries can be supported. Last but not least, our
data warehouse design considers other important dimensions
which enable complex analysis on baggage tracking data.

III. RFID-BASED AIRPORT BAGGAGE HANDLING

In airport baggage management a bag needs to pass different
stages to go from origin to final destination. Suppose that
Lisa needs to travel from Aalborg Airport (AAL) to Arlanda
Airport (ARN) via Copenhagen Airport (CPH). First, Lisa has
to check-in and handover her bag to the check-in desk staff.
Then the staff puts the bag into the conveyor belt for automatic
baggage sortation system. After passing all the stages inside
AAL, the bag is loaded into the aircraft using belt loader for
the targeted flight. As the bag has to be transferred to ARN,
upon arrival at CPH it is shifted to the transfer system. After
all the required stages at CPH, the bag is loaded to the aircraft
for its next flight to ARN. After arriving at ARN, the bag is
shifted to arrival belt and finally Lisa collects the bag from
the arrival belt.

Figure 1 shows an example of RFID reader deployment
at different locations of a baggage management system. An
RFID reader corresponds to the location where it is deployed.
For example, reader1 in Figure 1 corresponds to check-in1,
reader6 corresponds to Gateway-1 etc. The circles represent
the RFID reader and their activation range.

At check-in, an RFID tag is attached to the bag. The tag
is then read by readers is passes through their respective
activation ranges. An RFID reader continuously detects the
tags, and the reader’s controller software determines which
actual records are stored: if the read rate is 1 sec, an object
staying under a reader for 30 sec will generate 30 records,
unless it is specified that only one record can be sent per 15
sec, which will yield only 2 records.

An RFID tag has small built-in memory that stores bag
information. An example of this data is \{0123456789, 28APR,
AAL, SK1234, 28APR, CPH, LH2345, 28APR, ARN\}. The first
10 digits are the LicensePlate\(^2\). Specifically, the 1st digit is a
flag, the 2-4th digits state the bag issuer code (e.g., 117 for
SK (SAS)) and the 5-10th digits are the baggage tag number.
Next, the flight date is 28APR, the tag is printed in AAL(borg),
the first flight leg is SK1234 on 28APR to CPH, the second
leg is LH2345 on 28APR to ARN.

Due to the huge number of taggings, the 10 digit integer
LicensePlate is reused after some time. Thus, a BagID
is added to uniquely identify a bag. The records are stored in
the format: \{BagID, L, T, info\}, meaning that a reader at
location L detects a bag with ID BagID at timestamp T and
the tag stores the information info. Considering only location
and time related information, some example tracking records
are shown in Table I. The records are represented in the form:
\{ReadingID, BagID, LocationID, ReadingTime\}.

\(^2\)The license plate is a unique 10 digit code encoding bag information.
IATA specifies the rules for using the (Baggage) License Plate in Resolution
740B of their Passenger Services Conference Resolutions Manual.
TABLE I: Raw Baggage Tracking RFID Data

<table>
<thead>
<tr>
<th>(ReadingID, BagID, LocationID, ReadingTime)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(r1, b1, L1, 1) (r2, b2, L2, 2) (r3, b1, L3, 3) (r4, b1, L3, 4) (r5, b2, L3, 5) (r6, b2, L3, 6) (r7, b1, L4, 8) (r8, b1, L4, 9) (r9, b2, L4, 9) (r10, b2, L4, 10) (r11, b2, L5, 14) (r12, b2, L5, 15)</td>
</tr>
<tr>
<td>(r13, b1, L4, 19) (r14, b1, L4, 20) ...</td>
</tr>
</tbody>
</table>

As explained earlier, the raw readings contain many redundant records. A LocationTrace table can be constructed from the raw tracking sequence after eliminating the multiple readings. The format of the records in LocationTrace table is: (recordID, BagID, LocationID, t_in, t_out), where recordID is the identifier of each location trace record and t_in, t_out respectively represent the timestamps of first reading and last reading of BagID by the RFID reader deployed at location LocationID. This means that the bag was within the readers activation range during t_in, t_out. An example of a table containing location trace records from Table I is shown in Table II. In this table the record rec3 represents a bag b1 is observed by RFID reader at location L3 from time t4 to time t5, and record rec5 means that b1 is observed by reader at location L4 from time t5 to t6. This is a lossless compression with huge data reduction in volume.

TABLE II: Tracking records after duplicate elimination

| ObjectID | TrackingRecord (RecordID, ObjectID, LocationID, t_in, t_out) |
|-------------------------------------------------|
| b1 (rec1, b1, L1, 1) (rec3, b1, L3, 4, 5) (rec5, b1, L4, 8, 9) (rec8, b1, L4, 19, 20) | b2 (rec2, b2, L2, 2, 3) (rec4, b2, L3, 5, 6) (rec6, b2, L4, 9, 10) (rec7, b2, L5, 14, 15) ... |

IV. DATA WAREHOUSE DESIGN

To support business intelligence analysis using complex analytical queries, we propose a data warehouse design, shown in Figure 2. The proposed multi-dimensional data warehouse schema is a mixture of a star and a snowflake schema with one fact table and eight dimension tables. A star schema is less complex compared to a snowflake schema where the dimensions are partly normalized, but in some places, a snowflake schema is needed, as discussed in the following. In the following, the different dimensions are described, after which the fact table is introduced to link all the dimensions to the facts and calculated measures. As seen in the Figure 2, all the dimension tables follow the standard data warehouse convention of having auto-generated surrogate keys.

A. Dimension Descriptions

The different dimensions of the data warehouse presented in Figure 2 are described in this section.

1) Date and TimeOfDay Dimension: As seen in Figure 2, date and time has been split up into two dimensions in order to save records compared to a combined dimension table. If date and time were modeled in one table there could be over 86400 records for each day. In a year that would give over 31 million records in the dimension which could result in a slow query performance. Another reason for splitting up the two dimensions is that queries are normally performed on either a date basis or a time of day basis.

The main hierarchy of the Date dimension consists of: Year, Month, and DayOfMonth. To allow more detailed analysis, the dimension includes the following attributes: HalfYear, Quarter, WeekNumberOfYear and DayOfYear. DayOfWeek is included as a one level hierarchy to specify the specific days of the week. The HalfYear attribute indicates if it is the first or last half of the year. The DayOfWeek attribute enables queries on a specific day or range of days (1..365). The WeekNumberOfYear attribute enables queries on specific weeks of a year. The DayOfWeek makes it possible to query on a specific weekday and the last attribute DayOfWeekType enables classification of days, e.g. holiday, weekend, weekday, etc.

Location based events may occur on a particular date. For example Aalborg, Denmark celebrates a special occasion...
called Aalborg Carnival which is one of the biggest carnivals in Europe. As a result lot of people from Europe come to Aalborg to celebrate this occasion. Another example can be a strike in the city of an airport or conveyor belt broken of an airport etc. Additionally, bad weather in the area of an airport should be captured to relate baggage handling quality with the weather. So, it is very important to localize each date to the airport for capturing these types of location based special events and weather status. We use two special attributes: SpecialEvent, EventType which allow queries on a specific event or type of event. For localizing each date and date related information to a particular airport, we use AirportID as an attribute of the Date dimension. As a result the Date dimension has to store a copy of same date for each airport. Figure 4b shows the hierarchy of the Date dimension.

The Date dimension stores a copy of all stored dates for UTC which is independent of the airport. We store a default AirportID, (AllAirports), which indicates all airports and the other attributes also store default values for the corresponding UTC date. An example of date localization concept is shown in Fig. 5a. From Fig. 5a we can see that date May, 29, 2012 is localized for each airport and as a result it is possible to capture an special event i.e., ConveyorBeltBroken at AAL airport.

The Time dimension called TimeOfDay consists of the three necessary attributes to specify a time hierarchy down to a specific second: Hour, Minute, and Second. Attribute TimeOfDayType indicates whether this is rush hour or normal hour, and DaySections3Hours divides the 24 hours of a day into eight 3-hour sections. As a result it will be easier to analyze the baggage movement at different parts of the day. Figure 3b shows the hierarchy of the TimeOfDay dimension.

2) Location Dimension: In order to refer to different points of interest in the airport, a hierarchy of seven levels is introduced to handle the various degrees of detail. The lowest level in the hierarchy is denoted as a TagReaderLocation, which describes the specific location of an RFID reader like Check-in-1, Sorter-1, Sorter-2, Gateway-1, Arrival-1 etc. A location belongs to a category e.g., Sorter-1, Sorter-2 belongs to the location category Sorter. A number of location categories are grouped into an airport. The other levels of hierarchy of Location dimension are shown in Figure 4a.

The location hierarchy enables queries for the baggage movement at different abstraction levels, e.g., from one tag reader location to another tag reader location at the lowest level, and from one region to another region at the top level. Here, the meaning of region indicates the baggage movements between North America, Europe, Middle East and Africa (EMEA), Latin America (LATAM) and Asia-Pacific etc.,. The Location dimension is snow-flaked at airport level and we use Dim_airport as an outrigger dimension [7] as some other dimensions like Dim_Date, Dim_Flight etc., depend on the location at the airport level not at the tag reader level.

3) Status Dimension: The Status dimension captures the status of a bag while moving from one location to another. The status domain is \{WrongDestination, OK, LeftBehind, TrackedInNonRouteAirport, StillTrackedInNonRouteAirport, LongerDurationThanExpected, UnexpectedReader\}. To make a hierarchy, another column StatusGroup is introduced which contain the group of the status, e.g., WrongDestination, LeftBehind, TrackedInNonRouteAirport belong to mishandled group etc. Figure 3a shows the hierarchy of Status dimension.

4) Bag Dimension: The Bag dimension contains the identifier of the bag, LicensePlate and other planned travel information of bags like route, airline, date of departure etc. To keep track of the bags planned to move together in the same route and the same flight on the same date, we create a separate ID called BagFlightRouteID. This ID is generated while loading the data from the source schema. As a result, it creates a hierarchy in the Bag dimension. Each BagFlightRouteID corresponds to a given combination of StartDateID, FlightString and RouteString. Here, StartDateID is the departure date, FlightString contains a sequence of flights, e.g., "SK1202-QI1354-#", RouteString contains a sequence of airports of the route, e.g., "AAL-CPH-ARN-#". In both strings '#' indicates the end of the sequence. Use of these strings enables queries on flight and route sequences more easily. For example, to find the number of mishandled bags starting from AAL and
ending at ARN, the route string wild-card “AAL%ARN:” can be used. Figure 3c represents the hierarchy of Bag dimension.

5) Flight Dimension: In addition to the general information of a flight like FlightID, Airline, FlightNumber, departure date and time etc., we also store the source airport, destination airport, delay in departure, delay in arrival etc. Storing this delay information enables queries that can give an idea about how baggage mishandling is related with punctuality of flights. We store FlightNumberString which includes airline and flight number together e.g., “SK1202” indicates a flight of SAS (SK) Airline with flight number 1202. Instead of querying only on a flight number it is common to query on a flight based on FlightNumberString which can be rolled up to the airline level. Hierarchy of Flight dimension is represented in Figure 5b.

6) Handler Dimension: While traveling from origin to destination a bag is handled by many handler organizations at different stages of baggage movement. So handler is an extremely important entity related to baggage management. The Dim_Handler table represents the Handler dimension. It contains attributes HandlerID and Handler. The Handler dimension enables to find relationship between baggage handling quality and handlers.

B. Many-to-Many Relationship Between Flight and Bag

A bag can travel in many flights and a flight can carry many bags, and therefore the relationship between bag and flight is many-to-many. Additionally a sequence number is maintained when a bag is planned for traveling more than one flight. It also includes a transit duration i.e., duration between scheduled departure time of next flight and actual arrival time of the current flight. If a flight is the final flight of a bag in its sequence of flights then the transit duration is remain null. A standard approach of handling many-to-many relationship is the use of a bridge table that use foreign keys from both entities [8] e.g., Bag-Flight-Bridge(BagID, FlightID, SequenceIndex, TransitDuration) where BagID and FlightID are foreign keys taken from Bag table and Flight table respectively.

However this type of implementation produces huge amount of data in the bridge table which results a low query performance due to join lot of data. In Figure 6 the bag-flight bridge table shows some examples of association between bags and flights. Even though bags B1, B2 and B6 follows same flights with same sequences, due to the database design they have to be inserted thrice. Moreover in the whole table there are 2 different collection of pairs for ⟨FlightID, Sequence⟩ i.e., {(F1, 1), (F2, 2)} → 1 and {(F1, 1)} → 2. The process of transformation this table is exemplified in Figure 6.

Specifically, we create a separate table called BagFlightRoute table. It contains an ID BagFlightRouteID for each distinct set of pairs. Also, we store the flight string and route string for each BagFlightRouteID. The features of these two strings are described in the earlier sub section. For each BagFlightRouteID the pairs and its related info is stored in the FlightRouteBridge table. Finally in the bag table each bag is assigned with its relevant BagFlightRouteID. It makes a one-to-many relationship between BagFlightRoute table and bag table. The discussed transformation reduces huge amount of rows which results better query performance.

C. Fact Table

The Fact_Stay is the fact table and it binds the tracking records to the relevant entries in the dimensions. The relations to the TimeOfDay and Date dimension are represented twice for each DateTime attribute from the source data. This is to store both the wall clock time, and the UTC time when considering tracking records from different sites, e.g. AAL where the standard time zone is UTC/GMT+1 hour, Los Angeles International Airport (LAX) where the standard time zone is UTC/GMT-9 hours etc. As an RFID reader does not cover the whole area of a location, the value of t_out in Table II does not necessarily give us the time when an object actually get out from a location. As in the airport baggage tracking scenario a bag moves from one symbolic location to another symbolic location sequentially, instead of storing this t_out we store the time_end in the fact table which specify when the object reaches another location.

Attributes FromLocation and ToLocation specify the baggage movement from one tag reader location to the next tag reader location in its path sequence. Usually time_start and time_end of a particular bag indicates the time_in at FromLocation and time_in at ToLocation, respectively. However for the final location (which means there is no reading of a bag after this location), FromLocation is same as ToLocation and time_end specify the last reading time of the bag at this final location which is time_out at FromLocation. The duration taken by a bag to go from FromLocation to ToLocation is stored in the Duration attribute. If FromLocation to ToLocation both are from same airport then the value of Duration also represents the time spent by a bag at FromLocation.

We store HandlerID for each movement of a bag to track the responsible handler of the bag for that specific location. If it takes longer than expected to transfer the bag between the tracked location or the movement of the bag is not correct, then it will be possible to know which handler actually handled that
section and this mishandling can be attributed to the quality of that handler. The status of a bag at each stay is stored in StatusID attribute. Storing both source and destination for each tracking record enables lot of opportunity for analysis of baggage flow from one location to another in an easy and efficient way. This technique also allows giving a status to each movement of a bag from one tag reader location to another. The ResponsibleFlightID column points to the Flight dimension to keep track which flight is responsible for the movement. It allows to find the baggage handling quality of different flights and airline. Additionally we store FlightLegID that points to Dim_FlightRouteBridge table. It helps to find how baggage handling quality affects for transit duration.

D. Cube Design

A multi-dimensional cube is built on top of the data warehouse described above. There can be various types of measures for answering different types of analytical questions on RFID baggage data. We give a few examples of analytical questions: a) Average time taken by bags to move from sorter to gateway in Aalborg airport, b) Total number of bags handled by Copenhagen airport in the Christmas holiday, c) Total number of bag sent to wrong destination from Gothenburg airport in weekends of January 2012, d) Maximum time taken by bags to go from check-in to sorter in Stockholm airport. To answer these queries we used the following measures.

**Duration, maximum duration, minimum duration, average duration.** The duration column itself is used as measure and also enable other relevant measures like maximum duration, minimum duration, total duration taken by objects to go from one location to another. The Duration is also used for a computed measure avg duration i.e., the average duration taken by objects to move between locations. These measures are semi-additive and should be used with appropriate dimension to get relevant results. Because this is not meaningful to find the average, maximum, minimum duration without considering the FromLocation dimension. Moreover finding the sum of duration regardless of Bag dimension is also not meaningful. The use of maximum duration measure is shown in the following example MDX query in Q1. It returns a matrix showing max time taken by all bags to go from one tag reader location to another at AAL. In the query the parameter 2 for descendants function indicates two levels down from the airport level i.e., the tag reader level.

**BagID distinct count.** The number of distinct BagIDs in the fact table and grouping this on interesting dimensions give lot of interesting results. For example, the number of bags traveled from one tag reader location to another, and one country to another or region to region, number of bags mishandled at different airports, number of bags traveled in different routes, etc. For example the following MDX query, Q2 shows the number of bags mishandled at different airports and the result is ascending order based on number of bag(s) mishandled.

| Q2. SELECT [Measures].[Bag id Distinct Count] ON COLUMNS,
| ORDER (NonEmpty([FromLocation].[IATACode].members), [Measures].[Bagid Distinct Count]) ON ROWS
| FROM [cube_name]
| WHERE DESCENDANTS([Dim Status].[Status Group].&[Mishandled], 0) |

**Fact stay count.** The number of stay records in the fact table can be used for many types of computed measures like percentage calculation, average calculation, etc. For example finding the average duration taken by bags to go from one tag reader location to another at Aalborg airport can be calculated by the following MDX query, Q3 where we use the measure Fact Stay Count.

| Q3. WITH MEMBER [measures].[avgDuration] AS '([Measures].[Duration] / ([Measures].[FactStayCount])
| SELECT DESCENDANT([FromLocation].[LocationHierarchy].[Airport].&[AAL], 2) on Axis(1),
| DESCENDANTS([ToLocation].[LocationHierarchy].[Airport].&[AAL], 2) on Axis(0)
| FROM [cube_name]
| WHERE [Measures].[avgDuration] |

**Incorporating Flight and Bag many-to-many relationship into the cube.** Incorporating many-to-many relationship into a cube needs extra care. Because improper relationship between the dimensions and measures produce wrong aggregation results. As mentioned earlier we store both planned route and actual route of bags. The actual route taken by a bag is found from the tracking records stored in the fact table Fact stay. On the other hand a long chain of relational tables are maintained between Flight and Bag table for planned route and flights. To get the aggregate results like the number of bags planned for traveling by particular flight or airline, the Dim_Bag table is used as fact table and a measure Bag Count is created on this fact. Here the dimensions to be used by the measure are Dim_Bag, Flight and BagFlightRoute. As Dim_Bag and Flight is related by the Dim_FlightRouteBridge table through BagFlightRouteID, an intermediate fact table is required to incorporate them in the cube. So the Dim_FlightRouteBridge table is considered as an intermediate fact table where this fact is related with the Flight table by FlightID and to the Dim_Bag table by BagFlightRouteID. Now the Bag Count measure of Dim_Bag has a many-to-many relationship in the cube through the intermediate fact table Dim_FlightRouteBridge. The implementation of this concept in MS Analysis Services can be seen in Appendix A-A. An example query on this relation can be total number of bags planned to be carried by each airline and this can be answered by the following MDX query.

| Q1. SELECT DESCENDANTS([FromLocation].[LocationHierarchy].[Airport].&[AAL], 2) on Axis(1),
| DESCENDANTS([ToLocation].[LocationHierarchy].[Airport].&[AAL], 2) on Axis(0)
| FROM [cube_name]
| WHERE [Measures].[Maximum Duration] |
V. ETL DESIGN

In this section, we present the Extract-Transform-Load (ETL) design for loading relevant data from source databases to the data warehouse. The design is shown in Figure 7.

First Dim_Location_Airport is loaded with the relevant airports as this table is independent of any other dimension. This table is also loaded with a default airport "AllAirports" and an airport "INVALIDAirport". The default airport is used as a reference from the Date dimension to handle UTC date and time. On the other hand, INVALIDAirport is referred when there are some records containing some invalid values or null values. The Dim_Location_TagReader is loaded with tag reader information from the source data. Here also we have a default tag reader UnknownReader which points to INVALIDAirport. The name of the tag reader is made self contained, e.g., check-in reader of Aalborg Airport of Denmark is stored as 'DK.AAL.Check-in'.

The Date dimension is preloaded with relevant years and the TimeOfDay dimension is preloaded with each second of a day. Both Date and TimeOfDay dimension is loaded for the airports where RFID readers are deployed and reading records are coming from these airports. There is a copy of Date for the default airport which is discussed earlier.

The flight table is loaded directly from the source data where date and time of departure and arrival is pointed to the Date and TimeOfDay dimension respectively. The departure airport and arrival airport points to the Dim_Location_Airport table.

The Bag, BagFlightRoute and Dim_FlightRouteBridge are loaded together at the same time from the source data. The detailed algorithm can be seen in Appendix A-B.

The fact table Fact_Stay is loaded by the tracking records and relevant data from the source schema. Some preprocessing like creating some views in the source data can make the loading steps much easier and faster. We create two database view for this purpose. One view is created inside data warehouse schema with structure: v_track_airport(BagID, LocationID, t_in, t_out, AirportID), where the columns are taken from location tracking records of Table II and Airport Table of source schema. The v_track_airport includes AirportID which helps to easily track, when actually a bag changed its airport as well as tracking the responsible flight of a bag for the tracked airport. The overall algorithm to load the fact table can be seen in Appendix A-C.

VI. EXPERIMENTAL RESULTS

We implemented the relational data warehouse in SQL Server 2008 R2 and the cube in SQL Server Analysis Services 2008 using MS Business Intelligence Development Studio 2008. The source data is stored in Oracle 11g. The ETL loads the source data from Oracle into the SQL Server DW and is implemented in C# with .Net framework 4.0. For advanced querying and visualization, TARGIT BI suite 2K11 is used. The experiments are conducted on a laptop with an Intel Core i7 2.7 GHz processor with 8 GB main memory. The operating system is Windows 7 64 bit. We use RFID-based airport baggage tracking data collected from 57 RFID readers deployed at 57 different baggage handling locations at 7 airports in 7 cities in 2 countries. There were 149K bags with 4M filtered RFID readings. There was a huge data reduction after conversion of the RFID readings into location tracking records where it becomes 483K records. The readings were taken between 15-Dec-2011 and 17-Apr-2012 and the bags traveled on 36K flights.

Example Analysis Results We have generated many analysis results from the cube using through TARGIT BI suite, some are described here. The bar chart of Figure 8a shows bagID distinct count per Status where the status is one of {OK, Left Behind, and WrongDestination}. This shows that 97% of the bags are OK, 2% bags sent to the wrong destination, and 1% is left behind. Figure 8b shows the value of bagID distinct count grouped by the dimensions {Status, and TimeOfDay}. The status is drilled down to Wrong Destination and the TimeOfDay is at the DaySectionsIn3Hours level. This shows that among the 2% bags sent to wrong destination, 31% of the mishandlings happened between 15 PM to 18 PM, i.e., improvement efforts should focus here.

Performance We tested the above MDX queries on the MS Analysis Services cube and produced equivalent SQL
queries for both of the source database and the relational data warehouse. To allow direct comparison, the source data was migrated from Oracle into SQL Server 2008. To observe MDX query performance, MDX Studio\(^3\) was used. To observe SQL query performance, SQL Server 2008 Profiler was used and the queries is executed in SQL Server Management Studio. Each query is executed 5 times and for each execution, the query editor software is restarted and the cache is cleared. Finally, the average execution time is reported in Figure 8c and 8d for each query.

Query Q1 and Q3 is reported together in Figure 8c as both of them works with duration between tag reader locations of Aalborg airport. For the source data the value of duration, FromLocation and ToLocation is calculated using complex SQL queries and some intermediate views on the filtered RFID readings. Due to the complexity some criteria e.g., the final location of an object is not considered (In DW FromLocation and ToLocation is same for the final location of an object). The results show that, for Q1, the CPU time on cube is 7 times faster compared to relational data warehouse and 2.3K times faster compared to source data. For the same query CPU time on relational data warehouse is 313 times faster compared to source data. Similarly for other queries, the results show that, the CPU time for all of the cases cube is significantly faster whereas the source database is significantly slower. The queries on relational data warehouse are also much faster compared to source database. The queries on the source database are very slow due to the huge data volume which is also not structured for analysis purposes. On the other hand, the proposed relational data warehouse is specially structured for analysis purposes and we also have reduced the data volume. Finally, the designed cube employs specialized MOLAP storage and has pre-computed aggregate results, which make the queries much faster. The differences between the execution times will become even larger for larger data sets.

VII. CONCLUSION AND FUTURE WORK

In this paper we proposed a data warehouse solution for analysis of RFID-based airport baggage tracking data. We designed DW, including a number of complex dimensions and measures, in order to provide insight into the baggage tracking data and the reasons of baggage mishandling. We localized each date to a particular airport to capture special events related to that airport. We handled the many-to-many relationship between Bag and Flight dimension effectively both in the relational data warehouse and cube the design. The proposed data warehouse concepts can be used in similar types of indoor tracking application.

Future work will aim to solve the challenges encountered during this work as well as planned work. One important aspect is to scale the solutions to handle data from thousands of airport over long time periods, along with more precise capturing of the bag movement. This will require developing new pre-aggregation techniques. Another important aspect is to develop native support for spatio-temporal sequences, e.g., flight sequences, within the DW and BI tools, in order to get both more seamless querying and better performance.

ACKNOWLEDGMENT

This work is supported by the BagTrack project funded by the Danish National Advanced Technology Foundation under grant no. 010-2011-1.

REFERENCES


\(^3\)www.sqlbi.com/tools/mdx-studio/
APPENDIX A

A. Implementation of Many-To-Many Relationship into Cube

Implementation of many-to-many relationship into cube in MS Analysis services is shown in Figure 9. The figure shows the dimension usage of SQL Server Business Intelligence Development Studio 2008 for the cube containing many-to-many relationship between Bag and Flight. For generating the figure, a Visual Studio add-in BIDS Helper\(^4\) is used. From the figure we can see that Dim_Bag is used as both fact and dimension. Two measure groups on Dim_Bag and Dim_FlightRouteBridge are created. For Dim_Bag measure group the Flight dimension is used as many-to-many relationship where Dim_FlightRouteBridge is used as intermediate measure group. The BagFlightRoute is in regular relationship with the Dim_Bag measure group. Here the dimension column is BagFlightRoute.BagFlightRouteID and measure group column is Dim_Bag.BagFlightRouteID. Both Flight and BagFlightRoute dimension is in regular relationship with Dim_FlightRouteBridge measure group.

For BagFlightRoute dimension the dimension column is BagFlightRoute.BagFlightRouteID and measure group column is Dim_FlightRouteBridge.BagFlightRouteID. On the other hand for Flight dimension the dimension column is emphFlightID and measure group column is Dim_FlightRouteBridge.FlightID.

Fig. 9: Incorporating Flight and Bag relationship into cube

B. Algorithm: LoadBag_BagFlightRoute_FlightRouteBridge

The algorithm for loading Dim_Bag, BagFlightRoute and Dim_FlightRouteBridge is shown in Algorithm 1. In the algorithm at line 1, all the bags information is stored into variable B in the form (BagID, LicensePlate, FlightDate, Priority) as BagID wise ascending order. It helps to track easily which bags are already inserted into the data warehouse. If any error/failure occurs at the mid step during loading, we can quickly know which bags are already loaded by checking only the last inserted BagID. Putting additional condition: Where BagID > Last inserted BagID in line 1 of the algorithm can avoid reloading of existing bags and decreased the loading time. For each bag in the source data the algorithm gets all the flight info and the sequence of flights the bag has planned to travel. To find out the existence of a BagFlightRouteID for the bag with exact same number of pairs of (FlightId, SequenceIndex) we use the concept of relational algebra for exact relational division \(^{[11]}\) or relational division without reminder. A general relational division does not consider exact number of pair. For example consider the data of table Bag-Flight Bridge of Figure 6. Let us assume in Bag-Flight Route Bridge table there be only one BagFlightRouteID exist which is 1: \(\{\langle F1, 1\rangle, \langle F1, 2\rangle\}\). To find out the existence of BagFlightRouteID for bag B3 in Flight Route Bridge table, the relational algebra expression for general division operation can be: \(\Pi_{BagFlightRouteID,FlightID,SequenceIndex}(FlightRouteBridge) = \Pi_{FlightID,SequenceIndex}(BagID = B3(BagFlightBridge))\). For example the operation: \(\{\langle 1, F1, 1\rangle, \langle 1, F1, 2\rangle\} ÷ \{\langle F1, 1\rangle\}\) results 1. It seems that trying to find out the existence of BagFlightRouteID for bag B3 by the given general division operation returns 1 which is not correct. Because BagFlightRouteID 1 contains more flight sequences and we need to get a BagFlightRouteID containing exactly the set of value pair(FlightID, Sequence) of bag B3 i.e., \(\{\langle F1, 1\rangle\}\).

Algorithm 1 LoadBag_BagFlightRoute_FlightRouteBridge()

1: \(B := \) Set of all bags from the source data in the form (BagID, LicensePlate, FlightDate, Priority) in bagid wise ascending order;
2: \(BR := \) Set of routes of all the bags in the form (Bagid, Flightid, SequenceIndex) from the source data;
3: for each record \(b_i \in B\) do
4: \(R := \) all the route info of bag \(b_i.Bagid\) from \(BR\);
5: \(U := \bigcup_{j=1}^{SizeOf(R)} (\sigma_{Flightid,Sequenceindex}(R_j))\);
6: result := ExecuteQueryInDataWarehouse (WITH pair AS ( SELECT b.BagFlightRouteID as BagFlightRouteID FROM Dim_FlightRouteBridge b LEFT JOIN pair p ON ( b.SequenceIndex = p.SequenceIndex AND b.FlightID = p.FlightID ) ) GROUP BY b.BagFlightRouteID HAVING count(CASE WHEN p.flightid IS NULL THEN 1 END) = 0 AND count(*) = ( SELECT count(*) FROM pair ) )
7: if result.RowCount ≠ 0 then
8: Insert into Dim_Bag( b.Bagid, b.Licenseplate, result.BagFlightRouteID, b.Priority )
9: else
10: Generate FlightString and RouteString from \(BR\) and flight’s source and destination information;
11: Add a new record to BagFlightRoute table with a new BagFlightRouteID. This can be done by using database sequence or auto generated identifier;
12: Add records to Dim_FlightRouteBridge table with all the values of \(U\) and the new BagFlightRouteID. For TransferDuration subtract actual arrival time of current flight sequence from the schedule departure time of next flight. For the final flight of a BagFlightRouteID put TransferDuration as 0 (Zero);
13: Insert into Dim_Bag(b.Bagid, b.Licenseplate, The new BagFlightRouteID, b.Priority )

To match the exact pairs it is important to count the number of pair(s) at divisor and divide it only with those dividend which contain the same number of pair(s) as the divisor. The concept of exact relational division is implemented by a complex SQL query shown in line 6 of Algorithm 1. Getting any row by executing this SQL indi-

\(^4\)http://bidshelper.codeplex.com/
icates there exist a BagFlightRouteID with the given set of pairs (U of line 5 in Algorithm 1). If the SQL returns any BagFlightRouteID for this SQL, it inserts the bag with the returned BagFlightRouteID into the Dim_Bag table (line 8). In contrast with that, if the SQL does not return any BagFlightRouteID then a new record is inserted into the BagFlightRoute table with a new BagFlightRouteID (line 11). In line 13 it inserts the bag with this new BagFlightRoute.

C. Algorithm: LoadFactStay

The algorithm for loading the Fact table Fact_Stay is shown in Algorithm 2. As mentioned in the paper we have used some views to make the steps easy. The structure of the views are: v_bag_flight(BagID, FlightId, sequenceIndex, FromAirportID, ToAirportID, ActualDepatureTimeUTC, FlightLegID) and v_track_airport(BagID, LocationID, t_in, t_out, AirportID). At the beginning of the algorithm the tracking records, distinct BagID and flight info is loaded into variables (lines 1-3).

Algorithm 2 LoadFactStay()

1: TR := Set of all tracking records from v_track_airport in BagID and t_in wise ascending order;
2: B := Set of distinct BagID in TR on BagID wise ascending order;
3: BF := Set of all flight records from v_bag_flight in BagID and SequenceIndex wise ascending order;
4: for each BagID b_i ∈ B do
5: F := All records for bag b_i from BF;
6: R := All tracking records for bag b_i from TR;
7: R_end := Last record of R; seq := 1;
8: for each record r_i ∈ R \ R_end do
9: fromLoc := r_i.LocationID; toLoc := r_i.LocationID + 1;
10: fromAirport := r_i.AirportID; toAirport := r_i.AirportID + 1;
11: time_start := r_i.t_in; time_end := r_i.t_out;
12: Duration := time_end - time_start; Status := "OK";
13: if fromAirportID = toAirportID then
14: Find the status of the movement and the responsible flight with the help of F. ;
15: else
16: if toAirportID ≠ toAirportID then
17: Find ResponsibleFlightID from F with the help of seq then
18: Status := "WrongDestination";
19: seq++;
20: Insert a new record into the Fact table with b_i, fromLoc, toLoc, StatusID for Status, Duration, ResponsibleFlight, DateID and TimeID of time_start, time_end for both UTC, fromAirportID and toAirportID;
21: Now for the last tracking record R_end,
22: fromLoc := R_end.LocationID; toLoc := fromLoc;
23: time_start := R_end.t_in; time_end := R_end.t_out;
24: Duration := time_end - time_start; Status := "OK";
25: Insert a new record into the Fact table with b_i, fromLoc, toLoc, StatusID for Status, Duration, ResponsibleFlight, DateID and TimeID of time_start, time_end for both UTC, fromAirportID and toAirportID;

For each bag the algorithm retrieves and determine necessary fields for each tracking record and insert it into Fact_Stay table (lines 4-27). For each bag the algorithm filters the records of the particular bag and keep them into variables (lines 5-6). As for the final tracking record of a bag the FromLocation and ToLocation is same in the Fact_Stay table, we separate that tracking record into a variable R_end (line 7). This record is handled separately for each bag (lines 22-27). For each tracking record of a bag except R_end (line 8), the algorithm determine the FromLocation and ToLocation (line 10), time_start and time_end (line 11), check whether the bag changes the airport or not (lines 13 and 16) and based on the information it determines the Status of that particular tracking record (lines 14 and 17). The variable seq is used to track how many airport is changed by the bag which helps to retrieve flight info for that particular sequence and also to determine the Status. For R_end the operations are very similar to the other records except the value of ToLocation (line 21) and time_end (line 22). After retrieving and determining the values of necessary fields a record is inserted into Fact_Stay (lines 20 and 25).