# TEMPORAL SUPPORT IN RELATIONAL DATABASES

Bernadette Marie Byrne
School of Computer Science
University of Hertfordshire
Hatfield, UK
b.m.byrne@herts.ac.uk

Prajwal Triyambakaaradhya
School of Computer Science
University of Hertfordshire
Hatfield, UK
p.triyambakaaradhya@herts.ac.uk

## ABSTRACT

*This paper examines the current state of temporal support in relational databases and the type of situations where we need that support. There has been much research in this area and there were attempts in the American National Standards Institute (ANSI) and the International Organisation for Standardisation (ISO) standards committees in the late 1990s to add an extension called TSQL2 to the existing SQL standard. However no agreement could be reached as it was felt that some of the suggested extensions did not fit well with the relational model, as well as being difficult to implement. TSQL2 was abandoned and since then vendors have added their own data types, and if we are lucky, operators too in an attempt to provide support. However, to novice students and database designers it is often not apparent why some temporal concepts are difficult to deal with in a relational database. In teaching these concepts to students we use a Case Study (based on a real example) which illustrates the problems of providing temporal support by using examples of the data types which could be useful to solve temporal problems and the operators which are necessary to provide this.*

**Keywords**

*Timestamp, interval, valid time, transaction time, SQL, Relational Model*

## 1. INTRODUCTION

Temporal database support was first proposed in the early 1990's with the use of extensions to the standard database Structured Query Language (SQL92) called TSQL2 (Temporal Query Language). Work progressed to the late 1990s with a draft proposal to the ISO Technical Committee with an attempt to incorporate these extensions into what is now known as SQL3 [3]. However agreement was never reached and the project was cancelled at the end of 2001 in favour of other extensions to SQL3 such as XML. Therefore, support for temporal databases is currently loosely defined within the database community and not consistent across database vendors with some implementing their own interpretation of how to handle time past and time future within a database.

In 1992 a white paper [1] was published proposing a full temporal language design, TSQL, which would provide the basis for temporal database research. TSQL consolidated the different approaches to temporal data modelling at the time and included proposals for Valid Time, Transaction Time and Timestamps. The following year in July the Language Design Committee was formed from members of the database community to draw-up a language specification for TSQL2. In March 1994, a preliminary language specification from the committee appeared in the issue Association For Computing Machinery Special Interest Group on Management Of Data Records. After revisions, a definitive version of the TSQL2 language specification [2] was published in September 1994 and included specifications for temporal extensions such as Valid Time support for the past and future. It also specified a new PERIOD data type and four operators that would work on Valid Time data. The final specification with 28 commentaries was made available the following month, electronically. After this, members of the temporal database research community worked to transfer the temporal constructs and concepts of TSQL2 to SQL3, termed SQL/Temporal. This resulted in a formal proposal to the US National Standards Body in January 1995 and was to become part 7 of SQL3.

In March 2005, a paper [4] was published that highlighted some of the serious concerns and flaws in theTSQL2 approach, namely, that Valid Time support was achieved with the use of hidden, or implicit, attributes (normal attributes being explicit), they were hidden visually and in terms of accessibility. Having hidden attributes contravened the main principles of relation databases - that all information in a database should be represented as relations (tables), and that all data within a table can be referenced using column names. The paper therefore argues that the TSQL2 approach is not a relational approach. Another main concern raised was in the area of backward compatibility, the paper terms it as 'Temporal Upward Compatibility', (TUC). One of the pre-requisites of TSQL2 was that it must be TUC, so that it could be used on previous or legacy systems. This was shown not to be the case as the hidden variables required new syntax to access and that queries could produce different results when hidden attributes were introduced. There were also concerns that not enough temporal operators had been specified – there were no equivalents to PACK and UNPACK (mentioned later) among others.

As a consequence of these disagreements there are no definitive standards in ANSI and limited commands in SQL:2003 relating to temporal support. This has meant that database vendors have gone their own way - some vendors have implemented TSQL2, some have in parts, some have implemented SQL/Temporal and others have not implemented any temporal extensions.

## 2. THE CASE STUDY

A Holiday Bike Hire Company (HBHC) hires out vehicles to holiday-makers and locals at a popular holiday resort. It has a large stock of motorbikes that it needs to manage and keep track of. There was an existing HBHC database application that handles the renting out and the return of vehicles together with the money transactions involved. This application manages the daily rental of its stock of vehicles, checking them in and out, recording details of the agent and address of customer (client), calculating payment, printing invoices, etc. but there is no facility to handle a future reservation. There will also be information on clients, vehicles, agents, payments and locations.

The company currently can take a reservation for a client on a vehicle but the details are recorded on a separate Excel spreadsheet and there is no connectivity between the spreadsheet and the application database so reservations cannot be taken into account by the application when checking out a vehicle. Human interaction is required to liaise between application and spreadsheet and each vehicle is manually checked against the spreadsheet for availability. This process is both time consuming and prone to error as often the spreadsheet, being separate, is not up-to-date, contains invalid data, e.g. dates, or has missing data, e.g. blank cells. It was a regular occurrence in the height of the season to have irate customers in the shop because a motorbike they were expecting was not available.

At the time of the reservation, client details are taken together with the details of the vehicle being reserved. Details include ID number, start and end dates of the reservation and the date of the reservation itself. When a reservation is being made, a deposit is taken and the full amount to be paid is recorded. Reservations can be made by clients on the premises or by electronic means and this detail can also be recorded. The application needs a facility where it can record all the details of the reservation and reserve a vehicle for future hire by a client. This application then needs to account for reservations when determining which vehicles are available for hire during the day-to-day rental process. (Interestingly the original designers decided to record the reservations separately on a spreadsheet as they did not recognise that a reservation was in fact the same entity as a booking, only in the future.)

For the purpose of the case study the smallest unit of hire (time point) is one day. This makes it simpler to handle examples without having to account for half days. If the application can be demonstrated to work for whole days then it can be extended to work for half days by taking the time part of a date into consideration. For a reservation period, reservation dates are inclusive at the beginning and exclusive at the end.

## 3. MORE TEMPORAL QUESTIONS

There are some problem questions which could not easily be answered with the existing system.
Questions such as: Which motorbikes will be available between the 15th and 28th of March taking into account reservations that have already been made? Which bikes will be available for part of the period and from/to what date? Which 125cc motorbikes are available for a certain period?

The following tables are proposed:-

**Motorbike** (Reg, Model_no*, Original_price, Hid, Engine_size)

**Motorbike_Model** (Model_no, Price_per_day, Price_per_week, Price_Per_month, Model_name, make)

**Motorbike_Booking** (MBooking_ID, Reg*, Cust_ID*, Full_amount, Res_date, Start_date, End_date, Deposit)

**Customer** (Cust_id, FirstName, LastName, Home_street, Home_town, Home_postcode, Hotel_city, Hotel_street, Hotel_town, Hotel_postcode, Hotel_Province, CountryOfIssue, Expiry_date, PhoneNumber)

A database designer without the knowledge of temporal problems could end up with a similar design. The main table of interest is, of course, the motorbike booking table. This will include current bookings as well as past bookings and future bookings (reservations). The deposit attribute was used for bookings made over the phone or over the internet where a 20% deposit was required to confirm the booking. This would be null for customers who walked in off the street for an immediate booking as they would pay up front.

## 4. TIME GRANULARITY AND OVERLAPPING DATE PROBLEM

Granularity is an integral feature of temporal data. In the holiday bike case the time granule of time is a day,

From the above design a sample data set in a relation is as shown below.

| | MBOOKING_ID | CODE | REG | CUST_ID | FULL_AM... | DEPOSIT | RES_DATE | START_DATE | END_DATE |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 19 | s1 | 109 | 18 | 195 | 50 | 05-JUL-10 | 07-JUL-10 | 13-JUL-10 |
| 2 | 4 | s1 | 110 | 3 | 234 | 80 | 11-JUN-10 | 10-JUL-10 | 18-JUL-10 |
| 3 | 5 | s1 | 111 | 3 | 234 | 80 | 11-JUN-10 | 10-JUL-10 | 18-JUL-10 |
| 4 | 7 | s1 | 111 | 5 | 375 | 125 | 21-JUN-10 | 10-JUL-10 | 23-JUL-10 |
| 5 | 13 | s1 | 111 | 9 | 391 | 130 | 25-JUN-10 | 07-AUG-10 | 21-AUG-10 |
| 6 | 9 | s1 | 112 | 6 | 375 | 200 | 21-JUN-10 | 10-JUL-10 | 23-JUL-10 |
| 7 | 17 | s1 | 113 | 15 | 208 | 65 | 30-JUN-10 | 19-JUL-10 | 26-JUL-10 |
| 8 | 6 | s1 | 117 | 4 | 211 | 70 | 19-JUN-10 | 03-JUL-10 | 10-JUL-10 |
| 9 | 11 | s1 | 120 | 8 | 312 | 100 | 22-JUN-10 | 16-AUG-10 | 27-AUG-10 |
| 10 | 8 | s1 | 120 | 5 | 375 | 125 | 21-JUN-10 | 10-JUL-10 | 23-JUL-10 |
| 11 | 12 | s1 | 122 | 8 | 312 | 100 | 22-JUN-10 | 16-AUG-10 | 27-AUG-10 |
| 12 | 1 | s1 | 130 | 1 | 650 | 200 | 11-JUN-10 | 07-AUG-10 | 28-AUG-10 |
| 13 | 10 | s1 | 130 | 7 | 45 | 20 | 22-JUN-10 | 28-JUN-10 | 29-JUN-10 |

Figure 1 Screen shot of Motorbike_booking table.

Now let us consider a case where a user wishes to check the availability of a particular bike with the registration number 130, from the code snippet below.

```
Select reg, start_date, end_date
From motorbike_booking
Where (('29-jun-10' between start_date and end_date)
Or ('07-aug-10' between start_date and end_date))
And REG = 130;
```

This gives the following result:-

REG START_DATE END_DATE

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

130 07-AUG-10 28-AUG-10

130 28-JUN-10 29-JUN-10

Since the user query was between 29-JUN and 07-AUG, the result set shows that the bike is already booked for dates chosen by the user, even though they are available for partial hire from 30-Jun to 6-Aug. Ultimately a company might end up losing money because of this design. We will call this problem the Overlapping Date Problem. The only way a customer can get lucky is when he inputs the range of dates when the motorbike is actually available.

```
Select reg, start_date, end_date
From motorbike_booking
Where (('30-jun-10' between start_date and end_date)
Or ('06-aug-10' between start_date and end_date))
And reg = 130;
```

Results in the script output:

REG START_DATE END_DATE

---

NULL

The above result suggests there were no bookings made during the queried date a bike is available for hire. In the initial design it was assumed start and end date was enough to handle the booking, but even after applying arithmetic operations like “between” and “subtraction” it is not possible to pick the dates up. In order to eliminate the glitch one of the options would be to creating a row for every half a day of booking or day of booking depending on the time granularity. This requires the definition of another table to deal with time. This will work for any query because we have a row for each time granule and all the necessary calculations can be performed, however the problem with this is that if someone hires a bike for a month (which is often the case in the summer) we would have to have 31 rows in the booking table for a 31 day month, or 62 rows if we can rent the bikes out for half a day at a time. Obviously this has a knock on effect for performance. It might work for a small company such as HBHC but in a larger application would not be practical.

## 5. INTERVALS

An interval is a pair of dates plus the difference in between. If an interval is made up of a pair of dates, a start date and an end date, then the interval too is open to interpretation as to what it includes. If we look at the period starting at day 10 (d10) and ending day 15, what does this exactly include? It can be determined that d11, d12, d13 and d14 are definitely included but d10 and d15 may, or may not be, or a combination depending on the application. If d10 is included then the interval is said to be closed at its beginning otherwise it is said to be open at its beginning. Similarly, if d15 is included then it is closed at its end otherwise it is open at its end. Square brackets ‘[‘ and ‘]’ are used to signify a closed or included date at the beginning and end of the interval whereas round brackets ‘(‘ and ‘)’ signify an open date at the beginning and end of the interval. Combinations of these are used depending on situation. A colon ‘:’ is used to signify an Interval. Therefore, if the reservation periods are intervals that are closed at the beginning and at the end (inclusive), a reservation table can now be shown as Table 5. Alternatively, an inclusive start date but exclusive end date for {c1,r1} would be shown as [d07:d09).

Reservation

| Cust# | Res# | period |
|---|---|---|
| c1 | r1 | [d07:d09] |
| c1 | r2 | [d10:d15] |
| c2 | r3 | [d13:d14] |
| c2 | r3 | [d15:d16] |

Table 1 reservation table with intervals

The main consideration for temporal databases is this idea that a pair of dates and the period in between is treated as a value in its own right. This immediately gives advantages:

- It can be given its own data type.
- Queries become less complicated as the database engine would handle the start and end dates.
- One interval component is easier to handle than two dates and the bit in between.
- Comparisons between intervals are simplified because they are stored as values in their own right.
- Date ranges would not have to be explicitly tested for in queries
- Specific operators could be designed to handle specific temporal tasks.

# 6 Temporal operators

Additional operators have been suggested that can be applied to intervals so that the application does not have to explicitly manage them with query code.

### 6.1.1 OVERLAP

Used to determine if one interval overlaps another. In the HBHC case study it was often found that if a motorbike was not available for a period of hire because of demand then it was better to offer a motorbike that was available for part of the required period, either available for the initial part or later part of the period – the next best that could be offered. This operator could be used to determine such motorbikes.

### 6.1.2 CONTAINS

Used to determine if one interval is fully within another. In the HBHC case study, it would be useful to know whether there are reservations within the required period of hire or whether a bike is reserved for the whole time for which it is required

### 6.1.3 EQUALS

Used to determine if one interval exactly matches another

### 6.1.4 GREATER

Used to determine if one Interval is later/older than another. This operator together with Overlap could be used to determine if a motorbike is available for the later part of the hire.

### 6.1.5 LESSTHAN

Used to determine if one Interval is earlier/younger than another. This operator together with Overlap could be used to determine if a motorbike is available for the initial part of the hire.

### 6.1.6 PACK

A set operator that presents a table in such a way that no two intervals for an identifier either meet or overlap. It essentially condenses the table removing any redundancies yet is equivalent to the original.

| r# | period |
|---|---|
| r1 | [d05:d07] |
| r1 | [d08:d10] |
| r2 | [d01:d05] |
| r2 | [d03:d06] |

Table 2 Original table

| r# | period |
|---|---|
| r1 | [d05:d10] |
| r2 | [d01:d06] |

Table 3 Table packed

6.1.7 UNPACK

A set operator that presents a table in an expanded form where the intervals are decomposed into unit intervals. The expanded table, Table 4, is equivalent to the original, Table 2, and to the packed table, Table 3.

| r# | period |
|---|---|
| r1 | [d05:d05] |
| r1 | [d06:d06] |
| r1 | [d07:d07] |
| r1 | [d08:d08] |
| r1 | [d09:d09] |
| r1 | [d10:d10] |
| r2 | [d01:d01] |
| r2 | [d02:d02] |
| r2 | [d03:d03] |
| r2 | [d04:d04] |
| r2 | [d05:d05] |
| r2 | [d06:d06] |

Table 4 Table unpacked

The purpose of these last two temporal operators is to expand a temporal table into its constituent components where set operations such as MINUS, UNION, etc. can be performed before it is PACKed into condensed form.

6.1.8 COALESCE

With temporal tables it is sometimes very useful to bring together and concatenate date consecutive tuples that are equivalent in value. For example, when deciding which motorbikes to order for the following year, the HBHC want to find out the most popular motorbikes. One indicator would be to find the answer to the question: Which motorbikes are consistently continuously reserved and by how many days? Implying that demand is greater than supply.

Reservation

| v# | model | client# | rsv_strt_dat | rsv_end_dat |
|---|---|---|---|---|
| 120 | Scooter 125cc | 1388 | 01/03/11 | 13/03/11 |
| 130 | Moto 250cc | 1398 | 01/03/11 | 07/03/11 |
| 130 | Moto 250cc | 1440 | 07/03/11 | 14/03/11 |
| 130 | Moto 250cc | 1500 | 14/03/11 | 21/03/11 |

Table 5 Table of reservations

This question is very difficult to answer with SQL with the data in this format but if the table could be coalesced the SQL query would be much simplified.

Reservation2

| V# | Model | rsv_strt_dat | rsv_end_dat |
|---|---|---|---|
| 120 | Scooter 125cc | 01/03/11 | 13/03/11 |
| 130 | Moto 250cc | 01/03/11 | 21/03/11 |

Table 6 Coalesced table

The query is simplified together with any computational work that needs to be carried out.

```
SELECT v#,model,
rsv_end_dat – rsv_strt_dat AS ‘Consecutive Reservation’
FROM reservation2
ORDER BY 3 DESC;
```

# 7 VALID TIME AND TRANSACTION TIME

This is the time interval when an event is valid or active. It is the time that the event is believed to be true in the real world. For example, if motorbike number 160 has been reserved from 2nd April 2011 to the 14th April 2011, the valid time of the reservation will be the range between these two dates. Valid time can be inclusive or exclusive of start and end dates or a combination of each. The reservation start date is said to be time stamped at 2nd April 2011 and the end date time stamped at 14th April 2011. General convention seems to be that the start date is inclusive (open) and end date exclusive (closed).

| v# | rsv_strt_dat | rsv_end_dat |
|---|---|---|
| 160 | 02/04/2011 | 14/04/2011 |

Table 7

### 7.1.1 Transaction Time

This is the time when an event has been stored in the database – an event in the past. It is the time the event is true in the database. Here the reservation, numbered 296, for motorbike was entered onto the database on the 1st March 2011 at 9:00 am.

| r# | trans_time | Detail |
|---|---|---|
| 296 | 01/03/2011 9:00:00 | Inserted |

Table 8

### 7.1.2 Fixing the Past

With an OLTP database anything can be updated when the business requires it but with a temporal database where historical data is stored, are we changing history if this historical data is later updated? The answer is 'No', as a database is only a perception of the real world. Real world history (at the moment) cannot be changed but our perceptions certainly can change. For example, in our case study, if a reservation for a motorbike was made on the 1st March for a period from the 2nd April to 14th April but was later changed by the customer, on the 5th March, from the 1st April to the 13th April, the tuple will have to be updated accordingly. This is straight forward as the event is still in the future and merely an amendment. But what if the event had already occurred? Say, later on, on the 16th May, it transpired that the customer had in fact reserved the motorbike from the 3rd April to the 13th April. What should be done? Well, in this case the tuple would still need to be updated as it is only a representation of the real world event and the database needs to represent the real world as accurately as possible. We would only be updating our representation of history and not history itself.

In this example, the tuple representing the reservation start date would be initially time stamped 2nd April 2011 and the reservation end date time stamped 14th April 2011. The valid time would be the time span between the 2nd April and 14th April. The time the reservation is true. Transaction times are system generated and are not updateable by the user. Consequently they are often used in logs providing an audit trail of the sequence of events leading up to the current state. For the example above, Table 9, would be a log table for the tuple 123 showing the sequence of reservation changes.

| t# | trans_time | detail |
|---|---|---|
| 123 | 01/03/2011 09:00:00 | Inserted |
| 123 | 05/03/2011 10:00:00 | Updated |
| 123 | 16/05/2011 09:35:00 | Updated |

Table 9 Transaction times

Valid time and transaction time are obviously useful for recording temporal issues as described above. Transaction time would be useful for HBHC in a case where an awkward customer keeps changing their reservation dates. By storing a transaction time as well as the reservation time we could see how many times the customer has changed their mind over the dates they want to hire. The booking number would remain the same. The valid start and end date would be the actual current dates the customer wants to hire the bike for. However valid time and transaction times are a different and separate issue from the overlapping date problem with the HBHC.

## 8. OVERLAPPING DATES

Both start and end dates can either be inclusive or exclusive. An inclusive start date means the period begins on, and includes, the start day whereas an exclusive start date means the period begins the day after the start date. An inclusive end date means that the period extends up to, and includes the end day whereas with an exclusive end date, the period extends up to but not including the end day. This is an important as different databases/systems have different conventions.

To illustrate the scale of difficulty without temporal support it was decided to try and answer these questions with a conventional SQL query script.

If we take the following question: Which 125cc motorbikes are available for the period 15th March to the 28th March 2011 and which motorbikes are available for part of this period, providing dates from which they are available or to what date they are available? Table 10 is a list of 125cc Motorbikes with their reservations.

| Numbk | numve | rsv_strt_dat | rsv_end_dat |
|---|---|---|---|
| 298 | 120 | 18-Mar-11 | 30-Mar-11 |
| 299 | 121 | 13-Mar-11 | 16-Mar-11 |
| 300 | 122 | 13-Mar-11 | 31-Mar-11 |
| 301 | 130 | 17-Mar-11 | 24-Mar-11 |
| 269 | 131 | 27-Aug-10 | 30-Aug-10 |
| 302 | 132 | 14-Mar-11 | 29-Mar-11 |
| 303 | 140 | 15-Mar-11 | 28-Mar-11 |
| 296 | 142 | 14-Mar-11 | 28-Mar-11 |

Table 10 Reservations for 125cc motorbikes

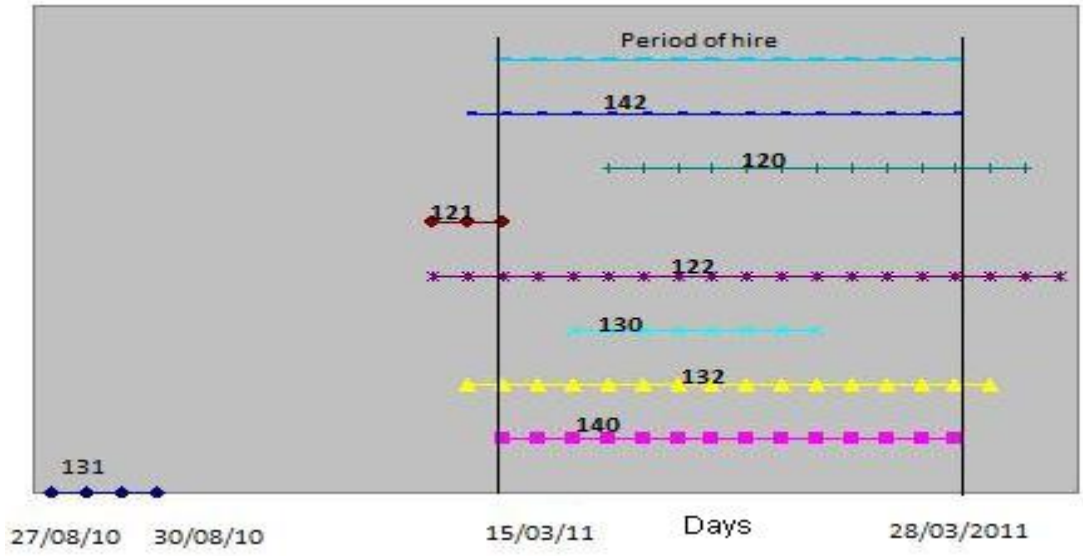


Figure 2. Representation of motorbike reservation test data

**Period equals a reservation**

Motorbike number (numve) 140. This motorbike has been reserved for exactly the required period and would not be available.

**A reservation is entirely within a period**

Motorbike number 130. This motorbike has been reserved within the required period and theoretically would be available for hire at the beginning until the 16th March and at the end of the required period from the 24th March.

**Period is entirely within a reservation**

Motorbike number 122. This motorbike has been reserved for more than the required period and would not be available for hire.

Motorbike number 142. This motorbike has also been reserved for more than the required period but its reservation ends on same date as the end date of the required period and would therefore not be available for hire.

Motorbike number 140.This motorbike has also been reserved for more than the required period but its reservation start date is the same as the required period's start date and so would also not be available for hire.

**Reservation overlaps the beginning of the period**

Motorbike number 121. This motorbike has been reserved at the beginning of the required period and would be available for hire from the 16th March for the later part of the period.

**Reservation overlaps the end of the period**

Motorbike number 120. This motorbike has been reserved towards the end of the required period and would be available for hire up to the 17th March.

**No overlap**

Motorbike number 131. This motorbike has no reservation infringement on the required period and would be available for hire.

Although the required results can be achieved with either PLSQL or with SQL (examples in the Appendices), the queries are complex and time consuming to formulate as every scenario of temporal interval infringement has to be catered for with SQL code. Date inclusion and exclusion have to be taken into account with SQL code. It requires extensive testing to ensure that the query performs as it should, especially around the date boundaries. A minor change to the specification would require a partial redrafting of code.

Therefore temporal functionality could be added to a conventional database application with the addition of date attributes to tables and that temporal processing can be handled by SQL without temporal extensions and we have examples of this in SQL (Microsoft Access) and in PLSQL. However, the queries are cumbersome and complicated to write and prone to error as minor details had to be manually catered for within the query. Obviously, inbuilt temporal extensions to SQL would make the job a lot easier and quicker to formulate.

# 9. TEMPORAL EXTENSIONS IN ORACLE

Oracle 11g does have some temporal extensions to its DBMS and SQL. The main ones were investigated with the view of incorporating or applying them to the case study tables.

### 9.1.1 The Interval Data Type

The DATE data type stores an instance in time whereas the INTERVAL data type stores a passage of time with granularity from years to seconds with two data types:

INTERVAL YEAR TO MONTH

INTERVAL DAY TO SECOND

Initially, this was thought to be a temporal extension that could be applied to the HBHC case study. But it soon transpired that although an interval of time could be stored, the start and end dates themselves were not included. The only way to store the reservation start and end dates was as in our original booking table. The reservation start and end dates together with the interval period could not be treated as one. Using this data type in our case study would see no advantages. Its main use is for adding an interval, day, week, etc. to a date.

### 9.1.2 The COALESCE Function

The COALESCE function was first thought to return tuples as described in section 6.1.8 However, the Oracle COALESCE function is used to return the first non-null expression working along a varying list of expressions of the same type. The COALESCE function would work for a Date data type but it could not be applied to tuples.

COALESCE (expr1, expr2, …..exprn)

COALESCE was not a temporal extension.

### 9.1.3 Timestamp Data Type

A Timestamp data type differs from a regular Date data type by its granularity. The Timestamp data type has a much finer granularity. The problem with time stamping with a Date data type is that if two events happened closer than one second apart the two events could be time stamped with the same date and time. The consequence being that a dense time dependent audit trail being incorrect or a sort based on timestamps being inaccurate. The Timestamp data type however, can store everything a Date data type can store but with an additional fractional second of '.000000'.

A regular Date data type can store: 15/03/2011 09:30:15

A Timestamp data type can store: 15/03/2011 09:30:15:000001

For the purpose of this project the granularity of the HBHC case study is one day so the Date data type is used for time stamping and the Timestamp data type offered no enhancement in this case.

## 10. CONCLUSION

Is the Holiday Bikes example a simple problem? No. On the surface and to a novice designer or student of databases this at first seems easy. The case study is useful for demonstrating the use of the simpler concepts of transaction time and valid time. Also it is useful for demonstrating that start date and end date attributes in themselves are not enough to solve the overlapping dates problem and that the one way to solve the problem is by having a row for each time granule in the booking table, but this is not always practical. However, as we have seen the problem of dealing with overlapping intervals in order to determine which bikes are free on certain dates is a non-trivial task. Mis-leading data type names such as “interval” in the Oracle DBMS are not helpful. At the end of last year a new edition of the SQL international standard SQL:2011 was brought out. It includes some new features in support of temporal data (mainly based on IBM’s DB2.) But once again the support is limited and further work is required in this area.

Also the case study is good for critical analysis in undergraduate and MSc projects using research questions such as:- Does an existing RDBMS offer sufficient temporal support? and Does this temporal support stray from the relational model of data?

These issues are obviously still current, and unresolved, as one of authors of the paper [4] has announced seminars [5] in 2011, one of which covers the subject of temporal databases and in the abstract of the seminar details suggests that there is a new approach to temporal database support and that it ‘fits squarely into the classical relational tradition’.

It has been over two decades since the first proposals for temporal support in the relational model. Yet most vendors have implemented very few temporal extensions. It might also be a case of impedance mismatch; where the relational model, which is based on set processing and single values at each row/column intersection would never support temporal aspects or its extensions.

## 11. REFERENCES


[1] Snodgrass, R., 1992. TSQL: A Design Approach
[2] Snodgrass, R. T. et al., 1994.TSQL2 Language Specification
[3] Snodgrass, R. T. et al., 1998. Transitioning Temporal Support in TSQL2 to SQL3
[4] Darwen, H., 2005. An Overview and Analysis of Proposals Based on the TSQL2 Approach
[5] Just SQL, 2011. Chris Date’s 2011 Seminars. Available at: http://www.justsql.co.uk/chris_date/chris_date.htm#3 [Accessed 06/05/2011]