Chapter 18
Active Data Warehousing

An *Active Data Warehousing* is where the data warehouse is immediately updated when the operational database is updated. The concept looks simple and attractive. However, there are a lot of complexities involved. This chapter will discuss the complexities thoroughly, covering (i) incremental updates, (ii) data warehousing schema evolution and (iii) operational database evolution. Firstly, the differences between Passive and Active Data Warehousing will be described.

18.1 Passive vs. Active Data Warehousing

A *Passive Data Warehouse* is a data warehouse that, once built, will remain unchanged. All of the data warehouses studied in the previous chapters are passive data warehouses. These are traditional data warehouses where a data warehouse contains star schemas that are subject-oriented, and the purpose of the data warehouse is decided at the design state. They are generally small and targeted star schemas. For example, a Product Sales star schema is for the sole purpose of analysing product sales and nothing else. A Computer Lab Activities star schema is for the sole purpose of analysing the use of computer labs by students. A Hospital Admission star schema is for the sole purpose of analysing patient admissions to hospitals. These are subject-oriented data warehouse, and the star schema is built for one specific purpose.

Although a data warehouse may contain a multi-fact star schema, it is still considered subject-oriented. Even in a multi-fact star schema, it only covers one subject, which is further divided into sub-subjects. For example, in the Domestic Cleaning Company case study, they employ cleaners to clean their clients' houses and offices. The star schema has two facts: one for Full-Time cleaners and the other for Part-Time cleaners, as they have different requirements, such as the number of hours they commit to the job, different payment rates and other conditions. Although

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2021

D. Taniar, W. Rahayu, *Data Warehousing and Analytics*, Data-Centric Systems and Applications, https://doi.org/10.1007/978-3-030-81979-8_18

they are separated into two facts, they have the same subject, which is recording their cleaning job activities (e.g. number of hours, number of cleaning jobs, payment, etc.).

Additionally, they are built based on historical data. For example, the Product Sales star schema only covers sales for a particular period, such as from 2010 to 2015; or the Hospital Admission star schema only covers patient admissions for a specific year, e.g. 2018. Once the data warehouse is built, there won't be any updates to the data warehouse. This is because the data warehouse is built based on historical data and for a specific time range. Hence, the data warehouse is not only subject-oriented but also non-volatile and time-variant. Non-volatile indicates that the data in the data warehouse will never change, whereas time-variant means that it is for a specific range of time, which is a past time period.

Data warehousing also provides various levels of granularity to support different levels of decision-making, where the upper levels precompute the fact measures from the lower levels. With the availability of multi-levels of star schema granularity, decision-makers can focus directly on a specific level of granularity of star schemas for more efficient data retrieval and subsequently more effective data analysis.

As Passive Data Warehousing is subject-oriented, targeted for a specific purpose, time-variant and non-volatile, the data warehouse is seen as a complete, self-contained and standalone package, once built. Decision-makers can then look into this package and analyse the data in it. As these packages are portable and the information in it is for a specific purpose, these packages are also known as *Data Marts*. In other words, the data is already in a self-contained package which is then saleable.

An *Active Data Warehousing* on the other hand is dynamic. When the operational database is updated, the data warehouse is immediately updated; hence, the data warehouse becomes "active". The data warehouse is always up-to-date, and it is no longer purely historical. Updates are also escalated to all levels of granularity of the star schema. The data warehouse is assumed to have direct access to the operational database, which is normally within the same system and environment. Figure 18.1 shows the architecture of Active Data Warehousing. The raw operational databases have been transformed into an integrated operational database, and the tables are directly accessible from (and available to) the data warehouse. Consequently, `create view` might be used in various levels of the star schema, which directly reflects any changes in the underlying tables, as views are virtual tables based on the underlying physical tables. Any changes in the underlying physical tables will automatically be reflected in the virtual tables. Additionally, certain database triggers can be used to automatically insert, update or delete the records in the data warehouse when an event (e.g. insert, update or delete) occurs in the underlying physical tables.

Designing an Active Data Warehousing adopts a bottom-up approach, where the star schema is built from Level-0, and the upper levels (e.g. Level-1, Level-2, etc.) are built on top of the immediate lower levels. Passive data warehouses may be built using the same approach, but as seen in the previous chapters, it is often easier to

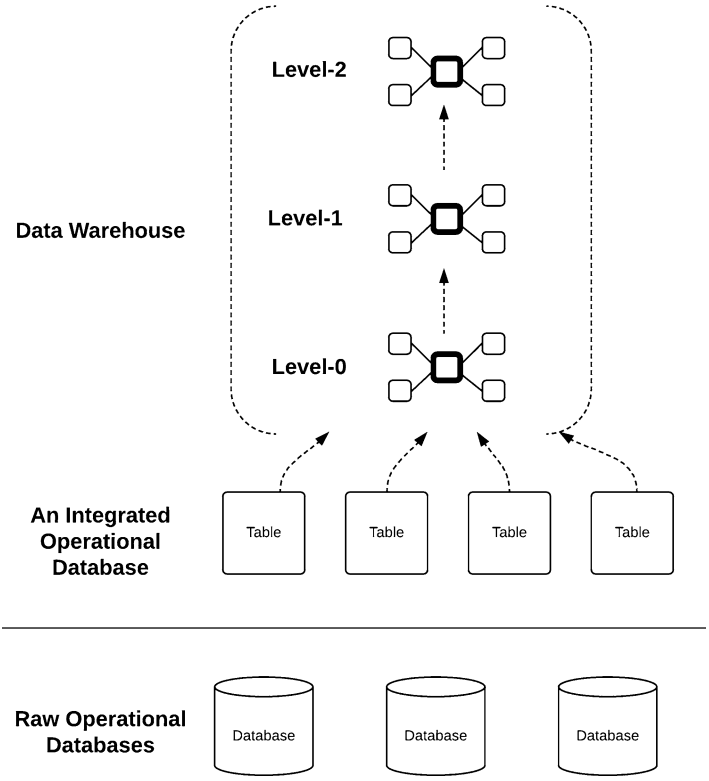

Fig. 18.1 Active Data Warehousing

build a data warehouse top-down, that is, to start from a reasonable upper level (e.g. Level-2) and move down level by level to the very bottom level (e.g. Level-0).

As this chapter focuses on Active Data Warehousing, we would like to examine what happens when Level-0 is updated in real time and what the impact is on the upper levels. This means that in Active Data Warehousing, updates are done automatically. This can be performed using virtual tables and database triggers to automatically execute the updates whenever needed.

Although most data warehousing is passive, some applications may benefit from Active Data Warehousing. This includes some monitoring sensors or real-time systems, where monitoring is done in real time using the current data, and hence, Active Data Warehousing is needed. Other systems that require continuous monitoring include air traffic control, the stock market, certain medical monitoring systems, etc.

18.2 Incremental Updates

Incremental updates in Active Data Warehousing are not to automatically update the data warehouse; there are other elements to consider, such as the "recentness" of the data in the data warehouse, as all data in the data warehouse need to be removed as they may not be as relevant as before. Hence, the data has an Expiry Date. Another element in Active Data Warehousing is that as time goes by, the data warehousing rules may have changed, so the data warehousing needs to adapt to the new rules.

18.2.1 Automatic Updates of Data Warehouse

Automatic updates refer to updates that occur immediately once the operational database has changed. The primary purpose of an operational database (or a transactional database) is to maintain new transaction records, as recent transactions keep coming to the operational databases. In technical terms, it is about the insertion of new records into the transaction table in the operational database.

We are going to reuse the Computer Lab Activities case study, discussed earlier in Chap. 14. The E/R diagram of the operational database is shown in Fig. 18.2. It is a simple database that consists of four tables: Lab Activities, Computer, Student and Degree. The data warehouse will focus on a fact measure that counts the number of usage (or number of logins) in the computer labs.

The data warehouse consists of three levels of granularity. The Level-0 star schema (or the lowest level), shown in Fig. 18.3, does not have any aggregation in the fact measure. The level of aggregation increases as the level of the star

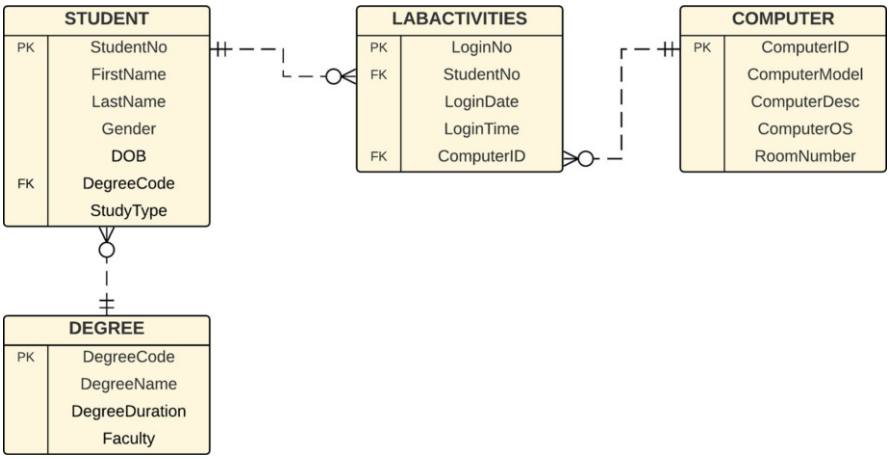

Fig. 18.2 E/R diagram for the Computer Lab Activities case study

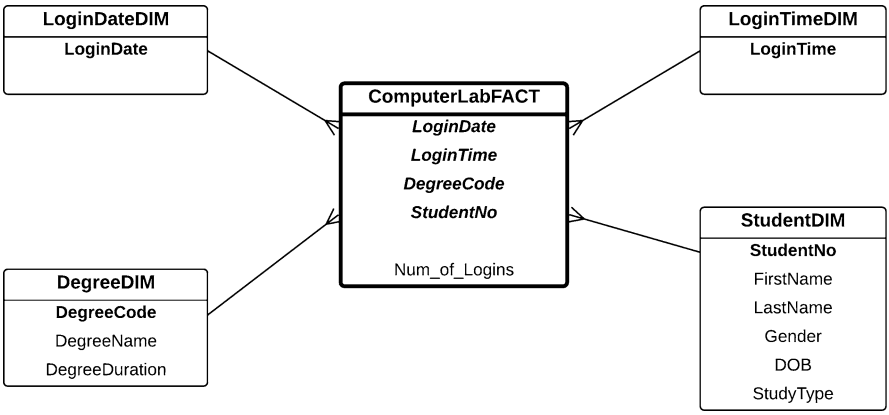

Fig. 18.3 Level-0 star schema for the Computer Lab Activities case study

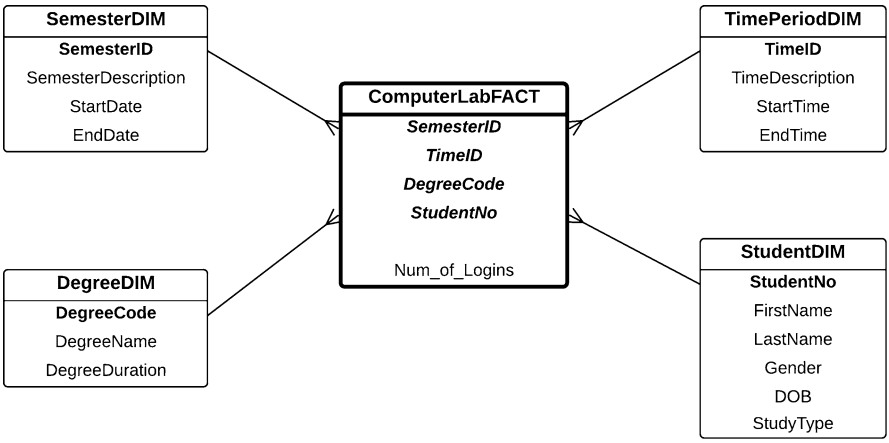

Fig. 18.4 Level-1 star schema for the Computer Lab Activities case study

schema increases. Figures 18.4 and 18.5 show the Level-1 and Level-2 star schemas, respectively.

The Level-0 star schema has four dimensions: Login Date, Login Time, Student and Degree Dimensions. These are the highest levels of granularity of data without any aggregation. Each record in the Fact Table refers to one login of one student at any particular date and time.

When the Login Date and Login Time are generalised to become Semester and Time Period Dimensions, the fact measure Number of Logins becomes aggregated. Hence, the granularity of the data is reduced. The star schema becomes Level-1. Note that the granularity is still determined by each individual student, but each student may log in multiple times during the semester, which is why the fact measure is now aggregated.

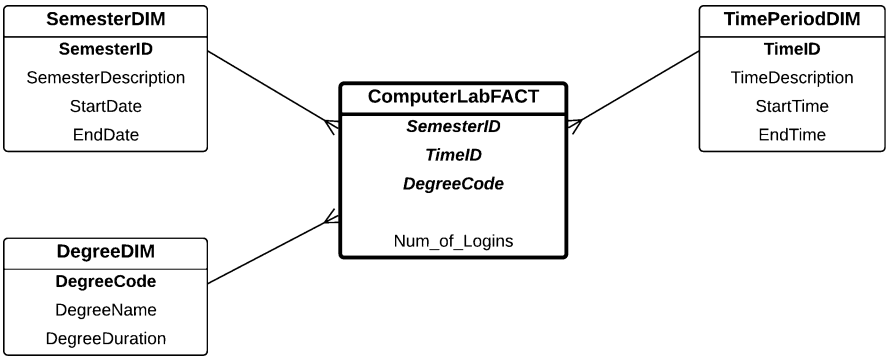

Fig. 18.5 Level-2 star schema for the Computer Lab Activities case study

Granularity is further reduced at Level-2 when the Student Dimension is removed from the star schema. As a result, the fact measure Number of Logins is no longer determined by individual students. The fact measure is now aggregated by Semester, Time Period and Degree Dimensions.

We will examine how the SQL commands in Active Data Warehousing are used to create (and maintain) the star schemas at different levels of granularity in data warehousing.

18.2.1.1 Level-0

The first step in star schema implementation is to create dimension tables, which is then followed by the TempFact and Fact Tables.

- **Dimensions**
  A common way to create dimension tables is by importing the necessary attributes and records from the relevant tables in the operational database. In this simple case study, only three tables from the operational database are used in the data warehouse, namely, Lab Activities, Student and Degree. Table Computer is not yet used in the data warehouse. So, for simplicity, any change to the Computer table will not affect the data warehouse.

```
create table LoginDateDim as
select distinct(LoginDate)
from LabActivities;

create table LoginTimeDim as
select distinct(LoginTime)
from LabActivities;

create table DegreeDim as
select DegreeCode, DegreeName, DegreeDuration
from Degree;
```

```
create table StudentDim as
select StudentNo, FirstName, LastName, Gender, StudyType
from Student;
```

Assuming that the operational database that contains the tables is within the same system, when new records are being added to the tables in the operational database, the dimension tables of the data warehouse must be updated automatically. Therefore, `create view` can be used to create the dimension tables, instead of `create table`.

```
create or replace view LoginDateDim as
select distinct(LoginDate)
from LabActivities;

create or replace view LoginTimeDim as
select distinct(LoginTime)
from LabActivities;

create or replace view DegreeDim as
select DegreeCode, DegreeName, DegreeDuration
from Degree;

create or replace view StudentDim as
select StudentNo, FirstName, LastName, Gender, StudyType
from Student;
```

If we would like to keep the dimension table as a table rather than a view (or a virtual table), we can use the `create table` command, but then we must have a database trigger that triggers an insertion of a new record. Take Student Dimension as an example. The trigger will trigger an insertion into the Student Dimension every time there is a new record inserted into the Student table in the operational database.

```
create table StudentDim as
select StudentNo, FirstName, LastName, Gender, StudyType
from Student;

create or replace trigger InsertNewStudent
  after insert on Student
  for each row
begin
  insert into StudentDim
  values (:new.StudentNo, :new.FirstName, :new.LastName,
    :new.Gender, :new.StudyType);
  commit;
end InsertNewStudent;
```

The advantage of using create view is that the update in the underlying table will be reflected immediately in the virtual table, so there is no need to have a separate trigger program that triggers an insertion to the dimension table.

For simplicity, we deal with the insertion of new records to the operational database only, that is, when new students log in to the Computer Lab Activities table or when there is a new degree program or when new students enrol. Update and Delete will be dealt with separately.

- **Fact**
  The Fact Table is usually created in this way:

```
create table ComputerLabFactLevel0 as
select distinct
  LoginDate, LoginTime,
  S.DegreeCode, L.StudentNo,
  count(L.StudentNo) as Num_of_Logins
from Student S, LabActivities L
where S.StudentNo = L.StudentNo
group by
  LoginDate, LoginTime,
  S.DegreeCode, L.StudentNo;
```

It is not possible to use `create view` to create the Fact Table, because the Fact Table needs an additional attribute which is not in the operational database. This attribute is the fact measure, which is the Number of Logins attribute. If we use fact without fact measure, we could simply use `create view` to create the Fact Table, as no additional attributes are required in the Fact Table. But with additional attributes, we need to use `create table` and then use a database trigger to automatically add a record to the fact when there is a new record inserted into the transaction table (e.g. Lab Activities table in the operational database).

```
create or replace trigger UpdateFactLevel0
  after insert on LabActivities
  for each row
  declare
    Degree Student.DegreeCode%type;
begin
  select DegreeCode into Degree
  from Student
  where StudentNo = :new.StudentNo;

  insert into ComputerLabFactLevel0
  values (:new.LoginDate, :new.LoginTime,
    Degree, :new.StudentNo, 1);
  commit;
end UpdateFactLevel0;
```

Whenever a new record is inserted into the Lab Activities table in the operational database, an automatic insert will be performed to the Fact Table. Note that since the Degree Code is in another table (e.g. Student Table), we must check the Degree Code from the Student table. A value of 1 is also inserted to the fact measure attribute in the Fact Table.

If we opt for Fact without fact measure, then `create view` is sufficient, and there is no need for a database trigger.

```
create or replace view ComputerLabFactLevel0 as
select distinct
  LoginDate, LoginTime,
  S.DegreeCode, L.StudentNo
from Student S, LabActivities L
where S.StudentNo = L.StudentNo;
```

In conclusion, dimension tables, which are directly obtained from the operational database tables, can use `create view`, but for the Fact Table, if it has fact measure attributes, the Fact Table must be created using create table but is automatically updated through an activation of the database trigger.

- **PK-FK Constraints**
  In Passive Data Warehousing, as described in all the previous chapters, there is no need to create a Primary Key-Foreign Key (PK-FK) constraint between each dimension key with the fact, simply because once the star schema is created and built, no records will be updated in the star schema (e.g. dimensions nor fact); there will be no insert as the data warehouse is passive and no delete. The PK-FK constraint in the operational databases is needed to maintain data integrity in the database because insert, update and delete of records are frequently performed and maintaining entity integrity and referential integrity is of the utmost importance. But in Passive Data Warehousing, since there is no insert, update or delete of records in the star schema, the need to have a PK-FK constraint is diminished, since there won't be any data integrity issues.

  However, in Active Data Warehousing, where the data warehouse is actively updated when there are changes in the operational database, the PK-FK constraint between dimensions and the fact becomes important to minimise data anomalies due to updates. Therefore, the PK constraint must be enforced in the dimension and the PK-FK constraint in the fact.

  If the dimensions in Level-0 are views from the operational database tables, it is very difficult to assume that the primary keys have been implemented correctly and primary keys cannot be built on top of views. Therefore, it is advisable that dimensions in the Level-0 star schema are created using the `create table` statement, and not through `create view`.

  After each dimension is created using `create table`, a PK constraint should be added. For simplicity, Login Date Dimension and Login Time Dimension are removed from the star schema (because they are one-attribute dimensions), and consequently, LoginDate and LoginTime attributes in the fact become non-dimensional keys. A PK constraint is only added to the Degree Dimension and Student Dimension.

```
alter table DegreeDim
add constraint DegreeDimPK primary key (DegreeCode);

alter table StudentDim
add constraint StudentDimPK primary key (StudentNo);
```

The Fact Table will have two FKs, each referencing these PKs. The Fact Table will also have a composite PK which combines all the four dimension keys.

```
alter table ComputerLabFactLevel0
add constraint DegreeDimFK foreign key (DegreeCode)
references DegreeDim (DegreeCode);

alter table ComputerLabFactLevel0
add constraint StudentDimFK foreign key (StudentNo)
references StudentDim (StudentNo);

alter table ComputerLabFactLevel0
add constraint FactPK
primary key (LoginDate, LoginTime, DegreeCode,
StudentNo);
```

If the fact is created using view (because it is fact without fact measure), it will not be possible to have FK and PK-FK constraints in the fact. In this case, there is no need to implement these constraints because the view is created using a join query with a distinct clause in the select, thereby ensuring that the results are unique.

18.2.1.2 Level-1

In the Level-1 star schema, the Semester Dimension replaces the Login Date Dimension, and the Time Period Dimension replaces the Login Time Dimension. The Semester Dimension contains only two semesters, whereas the Time Period Dimension divides the 24-h day into three time periods: morning (e.g. 6am–11:59am), afternoon (12pm–5:59pm) and night (e.g. 6pm–5:59am).

- **Dimensions**
  Semester and Time Period Dimensions are usually created using the following `create table` command, whereas the Degree and Student Dimensions may reuse the tables from the Level-0 star schema. Both Semester and Time Period Dimensions must be created manually, as they are not extracted from the operational database tables.

  There is no need to have a database trigger for these two dimensions because the current records are timeless, and no new records will ever be added unless there are rule changes about semesters, which will be covered in the next section. The records in the Time Period Dimension cover the entire 24 h, so there won't be any new records to be inserted in the future.

```
create table SemesterDim
(
  SemesterID            varchar2(10),
  SemesterDescription   varchar2(20),
  StartDate             date,
  EndDate               date,
  constraint SemesterDimPK primary key (SemesterID)
);
```

```
insert into SemesterDim
values ('S1', 'Semester 1',
  to_date('01-JAN', 'DD-MON'),
  to_date('15-JUL', 'DD-MON'));
insert into SemesterDim
values ('S2', 'Semester 2',
  to_date('16-JUL', 'DD-MON'),
  to_date('31-DEC', 'DD-MON'));

create table TimePeriodDim
(
  TimeID            varchar2(10),
  TimeDescription   varchar2(15),
  StarTime          date,
  EndTime           date,
  constraint TimePeriodDimPK primary key (TimeID)
);

insert into TimePeriodDim
values('1', 'Morning',
  to_date('06:00', 'HH24:MI'),
  to_date('11:59', 'HH24:MI'));
insert into TimePeriodDim
values('2', 'Afternoon',
  to_date('12:00', 'HH24:MI'),
  to_date('17:59', 'HH24:MI'));
insert into TimePeriodDim
values('3', 'Night',
  to_date('18:00', 'HH24:MI'),
  to_date('05:59', 'HH24:MI'));
```

- **TempFact**
  A TempFact Table is a temporary table that needs to be created before the final Fact Table is created because at least one of the dimension tables is created manually, not by exporting existing records from the operational database. The TempFact Table is usually created using the following command:

```
create table TempComputerLabFactLevel1 as
select distinct
  LoginTime, LoginDate,
  S.DegreeCode, L.StudentNo
from Student S, LabActivities L
where S.StudentNo = L.StudentNo;
```

  We could also make use of the Fact Table from Level-0 as the source for the TempFact Table on this level.

```
create table TempComputerLabFactLevel1 as
select distinct
  LoginTime, LoginDate,
  DegreeCode, StudentNo
from ComputerLabFactLevel0;
```

  As two new attributes need to be added to the TempFact Table to accommodate SemesterID and TimeID, we cannot use `create view` to create the TempFact

Table. The following commands add two new attributes to the TempFact Table and fill in the appropriate values.

```
alter table TempComputerLabFactLevel1
add (SemesterID varchar2(10));

update TempComputerLabFactLevel1
set SemesterID = 'S1'
where to_char(LoginDate, 'MM/DD') >= '01/01'
and to_char(LoginDate, 'MM/DD') <= '07/15';

update TempComputerLabFactLevel1
set SemesterID = 'S2'
where to_char(LoginDate, 'MM/DD') >= '07/16'
and to_char(LoginDate, 'MM/DD') <= '12/31';

alter table TempComputerLabFactLevel1
add (TimeID varchar2(10));

update TempComputerLabFactLevel1
set TimeID = '1'
where to_char(LoginTime, 'HH24:MI') >= '06:00'
and to_char(LoginTime, 'HH24:MI') < '12:00';

update TempComputerLabFactLevel1
set TimeID = '2'
where to_char(LoginTime, 'HH24:MI') >= '12:00'
and to_char(LoginTime, 'HH24:MI') < '18:00';

update TempComputerLabFactLevel1
set TimeID = '3'
where to_char(LoginTime, 'HH24:MI') >= '18:00'
or to_char(LoginTime, 'HH24:MI') < '06:00';
```

Since we need to use `create table`, we must then create a database trigger to automatically trigger an insertion into this TempFact Table when there is an insertion to the Fact Level-0.

```
create or replace trigger UpdateTempFactLevel1
  after insert on ComputerLabFactLevel0
  for each row
  declare
  TempSemesterID     SemesterDim.SemesterID%type;
  TempTimeID         TimePeriodDim.TimeID%type;

begin
  if (to_char(:new.LoginDate, 'MM/DD') >= '01/01')
    and (to_char(:new.LoginDate, 'MM/DD') <= '07/15')
    then
    TempSemesterID := 'S1';
  else
    TempSemesterID := 'S2';
  end if;
```

```
  if (to_char(:new.LoginTime, 'HH24:MI') >= '06:00')
    and (to_char(:new.LoginTime, 'HH24:MI') < '12:00')
    then
    TempTimeID := '1';
  elsif (to_char(:new.LoginTime, 'HH24:MI') >= '12:00')
    and (to_char(:new.LoginTime, 'HH24:MI') < '18:00')
    then
    TempTimeID := '2';
  else
    TempTimeID := '3';
  end if;

  insert into TempComputerLabFactLevel1
  values (:new.LoginTime, :new.LoginDate,
  :new.DegreeCode, :new.StudentNo, TempSemesterID,
  TempTimeID);

  commit;
end UpdateTempFactLevel1;
```

- **The Fact**
  After the TempFact is created, the final Fact Table for Level-1 can be created.

```
create table ComputerLabFactLevel1 as
select
  SemesterID, TimeID, DegreeCode, StudentNo,
  count(StudentNo) as Num_of_Logins
from TempComputerLabFactLevel1
group by SemesterID, TimeID, DegreeCode, StudentNo;

alter table ComputerLabFactLevel1
add constraint SemesterDimFK foreign key (SemesterID)
references SemesterDim (SemesterID);

alter table ComputerLabFactLevel1
add constraint TimePeriodDimFK foreign key (TimeID)
references TimePeriodDim (TimeID);

alter table ComputerLabFactLevel1
add constraint DegreeDimFK foreign key (DegreeCode)
references DegreeDim (DegreeCode);

alter table ComputerLabFactLevel1
add constraint StudentDimFK foreign key (StudentNo)
references StudentDim (StudentNo);

alter table ComputerLabFactLevel1
add constraint FactPK
primary key (SemesterID, TimeID, DegreeCode, StudentNo);
```

A database trigger must be created to ensure that for every record inserted into the TempFact Level-1, it will increase the fact measure Number of Logins by one.

```
create or replace trigger UpdateFactLevel1
  after insert on TempComputerLabFactLevel1
  for each row
begin
  update ComputerLabFactLevel1
  set Num_of_Logins = Num_of_Logins + 1
  where SemesterID = :new.SemesterID
    and TimeID = :new.TimeID
    and DegreeCode = :new.DegreeCode
    and StudentNo = :new.StudentNo;
  commit;
end UpdateFactLevel1;
```

18.2.1.3 Level-2

In Level-2, we basically remove the Student Dimension to lower the granularity of the fact measure Number of Logins. All other dimensions are reused. Fact Level-2 reuses Fact Level-1, where the aggregation of the fact measure is increased. There is no need to have a TempFact in Level-2 because we can create the fact in Level-2 by using `create view` from the fact in Level-1. As a result of this, there won't be any need for a database trigger.

```
create or replace view ComputerLabFactLevel2 as
select SemesterID, TimeID, DegreeCode,
  sum(Num_of_Logins) as Num_of_Logins
from ComputerLabFactLevel1
group by SemesterID, TimeID, DegreeCode;
```

18.2.2 Expiry Date

In Active Data Warehousing, data keeps coming to the data warehouse. As time goes by, the data warehouse collects a good wealth of data, which is beneficial for decision-making. However, the data also spans a long period of time. In some decision-making, decision-makers often want to focus on recent data, as old data (or very old data) in some applications may be meaningless, as the data has already expired. Additionally, if these data are still kept in the data warehouse, the analysis may be clouded by the existence of old data, as data warehousing often uses aggregate functions, which may aggregate all the data in the data warehouse. Therefore, in Active Data Warehousing, it is necessary to remove old data from the data warehouse. The old data should not and will not be removed from the operational databases because the operational databases should not only support day-to-day operations but also act as an archival repository.

In the Lab Activities case study, the star schema uses Semester as a Dimension, without the Year. The fact measure includes all login records. Hence, the measure can be outdated. For example, Number of Logins in Semester 1 for a particular course may include activities, which are simply too old to be considered (e.g. more than 5 years old). This then raises the notion of *Expiry Date* or *Expiry Time*, where the data input to the data warehouse are limited by the time, and as the time goes by, the old data become less important or even irrelevant, and in this case, they should not contribute to the data warehouse anymore. For example, we want to make the data warehouse active for the last 5 years only. Data older than 5 years old will be deleted from the data warehouse. So, this involves removing records, not inserting/updating records.

We are not removing old records from the operational database but removing old records from the Level-0 star schema. However, in the previous section, we have shown that there are two possible versions for Level-0 star schemas: one with fact measure and the other without fact measure. For the one with fact measure, `create table` is used, together with database triggers that trigger an insertion of records to the star schema when there are new records in the transactions (e.g. operational database). For the one without fact measure, we can simply use `create view`, which creates virtual tables from the operational database. These two scenarios will be discussed next.

18.2.2.1 Level-0

For the Level-0 star schema with fact measure, we can create a new Fact Table with a filtering condition on the Login Date (e.g. only to include the last 5 years' data).

```
-- ComputerLabFact (with fact measure)
drop table ComputerLabFactLevel0;
create table ComputerLabFactLevel0 as
select distinct
  LoginDate, LoginTime,
  S.DegreeCode, L.StudentNo,
  count(L.StudentNo) as Num_of_Logins
from Student S, LabActivities L
where S.StudentNo = L.StudentNo
and to_char(LoginDate, 'YYYY') >= '2015'
group by
  LoginDate, LoginTime,
  S.DegreeCode, L.StudentNo;
```

Unfortunately, this has to be done manually (including the FK and PK-FK constraints). Automatic deletion (or creation) can be done in other systems. There is no need to change the database trigger as the Fact Table is rebuilt. The database trigger is only used for the incremental insert of new records, which is not applicable in this case.

For the fact without fact measure, the `create view` command can be used. However, a filtering clause must be used. The filtering clause is used to select which records from the underlying operational database to use in the star schema.

```
-- ComputerLabFact (without fact measure)
create or replace view ComputerLabFactLevel0 as
select distinct
  LoginDate, LoginTime,
  S.DegreeCode, L.StudentNo
from Student S, LabActivities L
where S.StudentNo = L.StudentNo
and to_char(LoginDate, 'YYYY') >= '2015';
```

18.2.2.2 Level-1

The Level-1 star schema includes the creation of TempFact and Fact Tables. As described in the previous section, TempFact for Level-1 has been created using the `create table` command, together with a database trigger for insert (for incremental insert). When trimming old data from the data warehouse, we must execute the `delete` command manually.

```
delete from TempComputerLabFactLevel1
where to_char(LoginDate, 'YYYY') < '2015';
```

The final Fact Table was created using the `create table` command based on the TempFact Table. In other words, when the TempFact changes, we only need to activate a database trigger to trigger a delete to the final Fact Table.

```
create or replace trigger ReduceFactLevel1
  after delete on TempComputerLabFactLevel1
  for each row
begin
  update ComputerLabFactLevel1
  set Num_of_Logins = Num_of_Logins - 1
  where SemesterID = :old.SemesterID
    and TimeID = :old.TimeID
    and DegreeCode = :old.DegreeCode
    and StudentNo = :old.StudentNo;
  commit;
end ReduceFactLevel1;
```

18.2.2.3 Level-2

The Level-2 Fact Table is a virtual table from the fact in Level-1. So, when Level-1 is up-to-date, Level-2 will be up-to-date as well. In other words, in this example, no special maintenance is needed for the Level-2 star schema.

18.2.3 Data Warehouse Rules Changed

One of the important features of Active Data Warehousing is that it always contains up-to-date data. This means there is no time boundary for the lifetime of the data warehouse, except the Expiry Date discussed in the previous section. A consequence of having Active Data Warehousing is that the design that was developed in the beginning when the data warehouse was first built might have changed due to new business rules or new requirements. So, an important question that must be answered when dealing with Active Data Warehousing not only relates to an incremental update of new data but what happens if the rules have changed and what are the implications for the data warehouse.

Using the Lab Activities case study, Semester 1 is from 1 January to 15 July, and Semester 2 is from 16 July to 31 December. Suppose the semester dates change. Instead of Semester 1 being from 1 January to 15 July, it is now from 1 January to 30 June; hence, the two semesters are of equal length. The changes will only be effective from the new year, so this doesn't change the old data that is already in the data warehouse.

18.2.3.1 Level-0

Since the Semester Dimension starts to exist from Level-1 upward, the change of semester dates will not affect the Level-0 star schema as the Level-0 star schema does not have the Semester Dimension. The Level-0 star schema uses the Login Date Dimension when the actual date of the login is recorded.

18.2.3.2 Level-1

Two places that will be affected by the change of the semester dates: Semester Dimension and the TempFact.

- **Semester Dimension**
  For the Semester Dimension, the information about the semester dates is stored in the attributes, Start Date and End Date. These provide additional information about the semester, and they do not affect the accuracy of the fact measure. Dimension attributes, except the key, are considered additional information of the dimension and do not affect the fact in any way. The semester dates are only used when coding the TempFact. So, additional dimension attributes are not critical in terms of the fact.

  When the semester dates change, we can utilise a temporal dimension to store the history of the semester dates. Figure 18.6 shows a temporal dimension for the Semester, called Semester History Dimension. The Start Year and End Year attributes indicate the time period when the record is valid. For the current valid record, the End Year is recorded as either a null, or a maximum year of "9999",

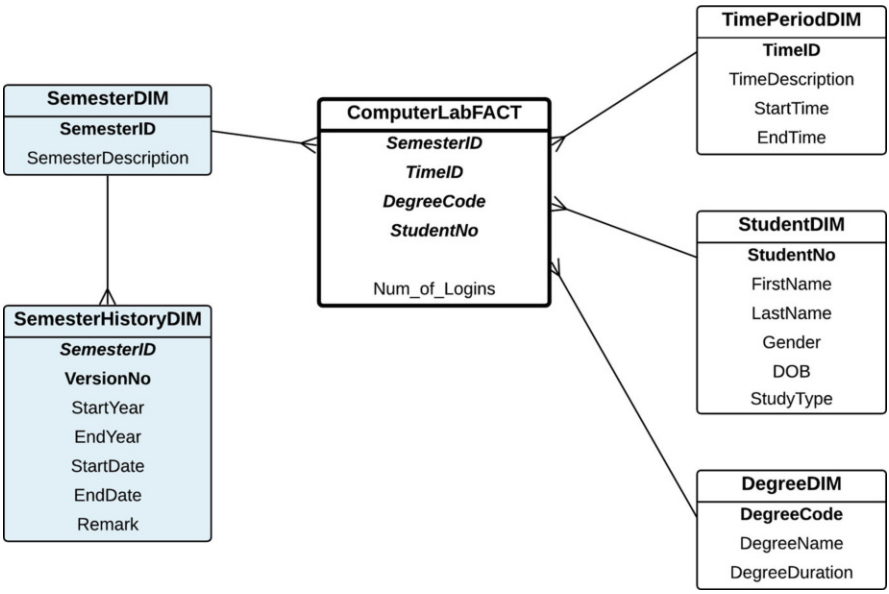

Fig. 18.6 Semester History Dimension

Table 18.1 Semester Dimension table

SemesterID	Semester description
S1	Semester 1
S2	Semester 2

Table 18.2 Semester History Dimension table

SemesterID	Version no	Start year	End year	Start date	End date	Remark
S1	1	2011	2020	1 January	15 July	
S1	2	2021	null	1 January	30-Jun	New semester dates
S2	1	2011	2020	16 July	31 December	
S2	2	2021	null	1 July	31 December	New semester dates

for example. The Start Date and End Date attributes store the semester dates, whereas the Remark attribute can be used to write any remark for the particular record. Tables 18.1 and 18.2 show an illustration of the Semester Dimension and the History Dimension tables. In this example, the old semester dates were from 2011 to 2020, and the new semester dates start from 2021 onward.

- **TempFact and Fact**
  The rule for the new semester dates will very much affect how the TempFact is maintained or updated. Since TempFact has been created, we only need to change the database trigger. Assuming that the new semester date rule starts in 2021, the only change will be checking the Login Date for SemesterID. The rest remains the same.

```
create or replace trigger UpdateTempFactLevel1
  after insert on ComputerLabFactLevel0
  for each row
  declare
  TempSemesterID     SemesterDim.SemesterID%type;
  TempTimeID         TimePeriodDim.TimeID%type;

begin
  if (to_char(:new.LoginDate, 'YYYY') >= '2021') then
    if (to_char(:new.LoginDate, 'MM/DD') >= '01/01') and
      (to_char(:new.LoginDate, 'MM/DD') <= '06/30') then
      TempSemesterID := 'S1';
    else
      TempSemesterID := 'S2';
    end if;
  else
    if (to_char(:new.LoginDate, 'MM/DD') >= '01/01') and
      (to_char(:new.LoginDate, 'MM/DD') <= '07/15') then
      TempSemesterID := 'S1';
    else
      TempSemesterID := 'S2';
    end if;
  end if;

  if (to_char(:new.LoginTime, 'HH24:MI') >= '06:00') and
    (to_char(:new.LoginTime, 'HH24:MI') < '12:00') then
    TempTimeID := '1';
  elsif
    (to_char(:new.LoginTime, 'HH24:MI') >= '12:00') and
    (to_char(:new.LoginTime, 'HH24:MI') < '18:00') then
    TempTimeID := '2';
  else
    TempTimeID := '3';
  end if;

  insert into TempComputerLabFactLevel1
  values (:new.LoginTime, :new.LoginDate, :new.DegreeCode,
  :new.StudentNo, TempSemesterID, TempTimeID);

  commit;

end UpdateTempFactLevel1;
```

The fact and the database trigger for the fact will not change because it will only affect new records and the semester date rule has been handled in the TempFact.

18.2.3.3 Level-2

The Level-2 star schema is unchanged because it is based on the Level-1 star schema, where the fact in Level-2 is created using the `create view` command.

The implication is that if Level-1 is updated correctly, the updates will be correctly reflected in Level-2 too. Hence, no special maintenance is needed for the Level-2 star schema.

18.3 Data Warehousing Schema Evolution

Active Data Warehousing is not only about automatic updates. The implication of having an active and dynamic data warehouse is more than just an automatic update when the operational databases are updated. Active Data Warehousing indirectly imposes some degree of interdependency not only between operational databases and data warehouses but also among the star schemas of different granularities within the data warehouse.

As Active Data Warehousing is time-boundless, the data warehousing requirements first used to design the data warehouse might have evolved over time. There might be a need to change the requirements. Consequently, data warehousing schema evolves over time. This is then known as *data warehousing schema evolution*. There are four types of data warehousing schema evolution (Fig. 18.7):

1. Changes to a star schema at one level propagating to upper levels,
2. Changes to a star schema at one level which do not affect the upper levels,
3. Inserting a new star schema into the data warehouse, and
4. Deleting a star schema from a data warehouse.

18.3.1 Changes Propagating to the Next Levels

In a data warehouse consisting of star schemas of various levels of granularity, when one star schema changes the structure, the changes may propagate to the upper levels of the star schema. In Active Data Warehousing, star schemas are reactive to changes, so change propagation must be dealt with in the next levels of the star schema. For example, if a Level-1 star schema changes its structure due to new requirements, the changes might be propagated to the Level-2 star schema and so on.

This change propagation only occurs in Active Data Warehousing, not only because of the interconnection between the star schemas in different levels but also due to the reactive nature of Active Data Warehousing, where updates, not only in the data but also in the schema or structure, are propagated immediately to the relevant star schemas.

In the Lab Activities case study, the Semester Dimension does not contain any information about the Year. This means that the analysis is purely based on semester, such as comparing the amount of computer usage between Semester 1 and Semester 2 for a particular degree, but it does not consider the year. This implies that the

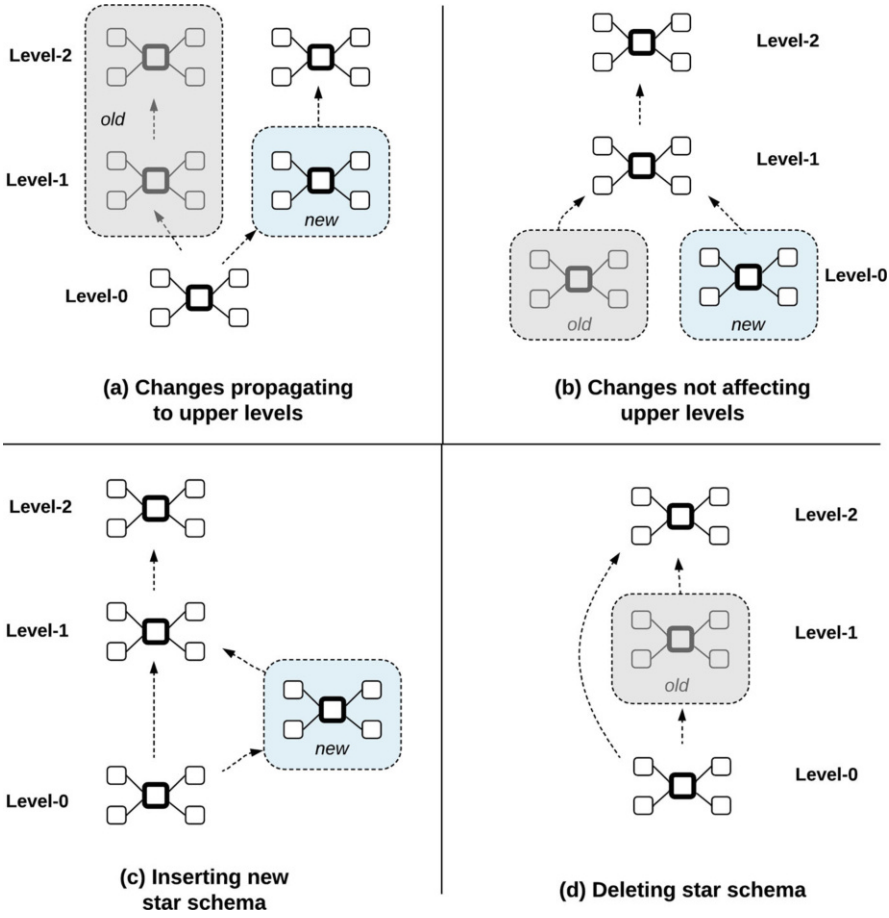

Fig. 18.7 Four cases of data warehousing schema evolution

comparison aggregates the data of all the years according to the semesters. Hence, old data from years ago will still be aggregated with the current data of the two semesters. In other words, the data relating to Semester 1 contains an aggregation of all Semester 1 data, regardless of the year. This kind of analysis will be skewed toward old data, which is why Expiry Date is used to limit the range of data in the data warehouse.

Another way to deal with this situation is by including the Year information into the semester. Therefore, there will be a separate record for each semester/year. In this way, the data analysis can be based not only on semester but also on semester/year, or only year. Adding the Year information into the requirement is an example where the data warehouse schema has evolved. We will see next how the additional requirement to data warehousing might change the schema and how it might propagate to the next levels.

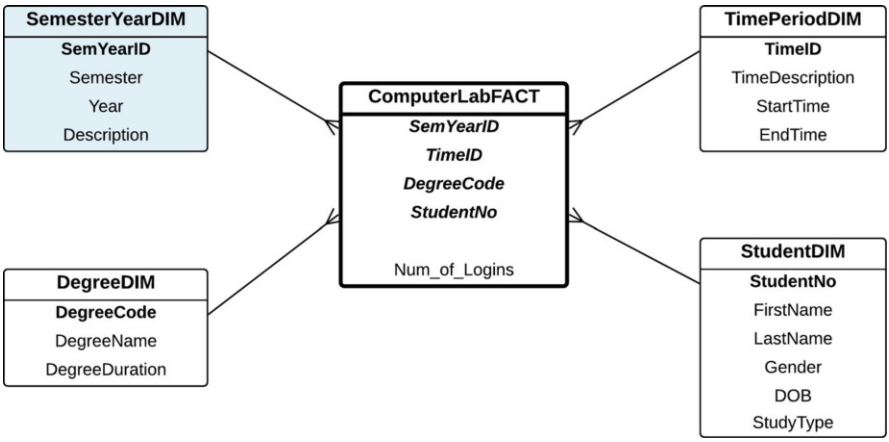

Fig. 18.8 Level-1 star schema with Semester Year Dimension

18.3.1.1 Level-0

The semester and year information will not appear until Level-1, so there are no changes to the Level-0 star schema. The Level-0 star schema stores records in their detailed form, including Login Date and Login Time, but they are not yet aggregated to the semester or year level.

18.3.1.2 Level-1

In the Level-1 star schema, the Semester Dimension is now changed to the Semester Year Dimension, incorporating the year information to the dimension. Other dimensions remain unchanged. Figure 18.8 shows the star schema with the Semester Year Dimension, where the SemYearID attribute is the surrogate key of the dimension. It also includes other attributes, such as Semester, Year and Description.

In the previous version, which included only the Semester Dimension, once the dimension is created and populated (with two records, Semester 1 and Semester 2), this dimension will never change, because any new transaction records can be categorised into either Semester 1 or Semester 2. But now, the dimension includes the Year information. Consequently, when there is a new transaction record which includes the new semester/year, a new record in the Semester Year Dimension must be recorded. In other words, we must now maintain not only the Fact Table of the star schema but also the Semester Year Dimension table.

- **Semester Year Dimension**
  The Semester Year Dimension table must first be created when the data warehouse is built. The Semester Year information will be converted from the Login Date attribute in the operational database.

```
create table SemesterYearDimTemp as
select distinct LoginDate
from LabActivities;

alter table SemesterYearDimTemp
add (
  SemYearID     varchar2(10),
  Semester      varchar2(2),
  Year          varchar2(4),
  Description   varchar2(20)
);

update SemesterYearDimTemp
set Semester = 'S1'
where to_char(LoginDate, 'MM/DD') >= '01/01'
and to_char(LoginDate, 'MM/DD') <= '06/30';

update SemesterYearDimTemp
set Semester = 'S2'
where Semester is null;

update SemesterYearDimTemp
set Year = to_char(LoginDate, 'YYYY');

update SemesterYearDimTemp
set SemYearID = Year || Semester;

create table SemesterYearDim as
select distinct SemYearID, Semester, Year, Description
from SemesterYearDimTemp;
```

After the dimension is created, the dimension must be maintained so that the new transaction records from the operational database will be immediately reflected in this star schema. The following is the process to maintain the Semester Year Dimension table. After converting the Login Date information from the operational database into a suitable format for the Semester Year Dimension table, it will first check if the record exists in the Dimension or not. If the record does not exist, the record is inserted into the Dimension.

```
create or replace trigger AddSemesterYear
  after insert on LabActivities
  for each row
  declare
    TempSemester       SemesterYearDim.Semester%type;
    TempYear           SemesterYearDim.Year%type;
    TempSemesterYearID SemesterYearDim.SemYearID%type;
    TempDescription    SemesterYearDim.Description%type;
    IsFound number;

begin
  if to_char(:new.LoginDate, 'MM/DD') >= '01/01'
  and to_char(:new.LoginDate, 'MM/DD') <= '06/30' then
    TempSemester := 'S1';
```

```
  else
    TempSemester := 'S2';
  end if;

  TempYear := to_char(:new.LoginDate, 'YYYY');
  TempDescription := null;
  TempSemesterYearID := TempSemester || TempYear;

  select count(*) into IsFound
  from SemesterYearDim
  where SemYearID = TempSemesterYearID;

  if IsFound = 0 then
    insert into SemesterYearDim
    values (TempSemesterYearID, TempSemester, TempYear,
    TempDescription);
    commit;
  end if;

end AddSemesterYear;
```

- **TempFact**
  A TempFact Table can be created in the usual way, but the Fact Table from Level-0 is reused.

```
create table TempComputerLabFactLevel1 as
select distinct
  LoginTime, LoginDate,
  DegreeCode, StudentNo
from ComputerLabFactLevel0;
```

Adding the SemYearID attribute into TempComputerLabFactLevel1 table can be tricky as Login Date needs to be converted to SemYearID.

```
alter table TempComputerLabFactLevel1
add (SemYearID varchar2(10));

update TempComputerLabFactLevel1
set SemYearID = to_char(LoginDate, 'YYYY') || 'S1'
where to_char(LoginDate, 'MM/DD') >= '01/01'
and to_char(LoginDate, 'MM/DD') <= '06/30';

update TempComputerLabFactLevel1
set SemYearID = to_char(LoginDate, 'YYYY') || 'S2'
where to_char(LoginDate, 'MM/DD') >= '07/01'
and to_char(LoginDate, 'MM/DD') <= '12/31';
```

Converting the Login Time attribute to TimeID attribute is done as shown in the previous section.

```
alter table TempComputerLabFactLevel1
add (TimeID number);

update TempComputerLabFactLevel1
set TimeID = '1'
```

```
where to_char(LoginTime, 'HH24:MI') >= '06:00'
and to_char(LoginTime, 'HH24:MI') < '12:00';

update TempComputerLabFactLevel1
set TimeID = '2'
where to_char(LoginTime, 'HH24:MI') >= '12:00'
and to_char(LoginTime, 'HH24:MI') < '18:00';

update TempComputerLabFactLevel1
set TimeID = '3'
where to_char(LoginTime, 'HH24:MI') >= '18:00'
or to_char(LoginTime, 'HH24:MI') < '06:00';
```

Finally, the database trigger to automatically trigger an insertion to the TempFact is as follows:

```
create or replace trigger UpdateTempFactLevel1
  after insert on ComputerLabFactLevel0
  for each row
  declare
    TempSemYearID
      TempComputerLabFactLevel1.SemYearID%type;
    TempTimeID
      TempComputerLabFactLevel1.TimeID%type;
begin
  if (to_char(:new.LoginDate, 'MM/DD') >= '01/01') and
    (to_char(:new.LoginDate, 'MM/DD') <= '06/30') then
    TempSemYearID := to_char(:new.LoginDate, 'YYYY')
    || 'S1';
  else
    TempSemYearID := to_char(:new.LoginDate, 'YYYY')
    || 'S2';
  end if;

  if (to_char(:new.LoginTime, 'HH24:MI') >= '06:00') and
    (to_char(:new.LoginTime, 'HH24:MI') < '12:00') then
    TempTimeID := '1';
  elsif
    (to_char(:new.LoginTime, 'HH24:MI') >= '12:00') and
    (to_char(:new.LoginTime, 'HH24:MI') < '18:00') then
    TempTimeID := '2';
  else
    TempTimeID := '3';
  end if;

  insert into TempComputerLabFactLevel1
  values (:new.LoginDate, :new.LoginTime,
  :new.DegreeCode, :new.StudentNo,
  TempSemYearID, TempTimeID);

  commit;
end UpdateTempFactLevel1;
```

- **The Fact**
  The final Fact Table for Level-1 is created by aggregating the four dimension identifiers and counting the StudentNo. Although the Semester Year Dimension has been created (and maintained using a database trigger) as previously explained, there is no guarantee that the new SemYearID has been inserted into the Semester Year Dimension table. Hence, it would have been easier if the PK-FK constraint between the fact and the dimension is not enforced to guarantee that the fact level will be created successfully. In the end, the Semester Year Dimension table will have all the semester year records anyway; however, we cannot guarantee that the Semester Year Dimension table has been populated with the new semester year record prior to the creation of this Fact Table.

```
create table ComputerLabFactLevel1 as
select
  SemYearID, TimeID, DegreeCode, StudentNo,
  count(StudentNo) as Num_of_Logins
from TempComputerLabFactLevel1
group by SemYearID, TimeID, DegreeCode, StudentNo;
```

  Once the Fact Table is created, to maintain this table, a database trigger must be used to ensure that when new records are inserted into the TempFact Table, they will be reflected in the Fact Table, as the fact measure Number of Logins needs to be incremented properly. If it is a new record in the fact (with the same dimension keys), the record will be inserted into the Fact Table. If the record already exists in the Fact Table (with the same dimension keys), the fact measure will simply be incremented by one value.

```
create or replace trigger UpdateFactLevel1
  after insert on TempComputerLabFactLevel1
  for each row
  declare
    IsFound number;
begin
  select count(*) into IsFound
  from ComputerLabFactLevel1
  where  SemYearID = :new.SemYearID
    and TimeID = :new.TimeID
    and DegreeCode = :new.DegreeCode
    and StudentNo = :new.StudentNo;

  if IsFound <> 0 then
    update ComputerLabFactLevel1
    set Num_of_Logins = Num_of_Logins + 1
    where SemYearID = :new.SemYearID
      and TimeID = :new.TimeID
      and DegreeCode = :new.DegreeCode
      and StudentNo = :new.StudentNo;
  else
    insert into ComputerLabFactLevel1
    values (:new.SemYearID, :new.TimeID, :new.DegreeCode,
    :new.StudentNo, 1);
```

```
    end if;
    commit;
  end UpdateFactLevel1;
```

18.3.1.3 Level-2

The changes in the Level-1 star schema, namely, the changes of the Semester Year Dimension, are carried forward to the upper level: Level-2. The changes are reflected at the star schema level, where the Semester Year Dimension is still used (refer to Fig. 18.9 for the Level-2 star schema). The Level-2 star schema basically removes the Student Dimension from Level-1.

From the technical implementation point of view, the Level-2 fact reuses the Level-1 fact, and in Level-1, the fact already takes care of the new Semester Year Dimension. Therefore, the impact to the Level-2 fact is technically minimal, which is shown in the following `create view` statement.

```
create or replace view ComputerLabFactLevel2 as
select SemYearID, TimeID, DegreeCode,
  sum(Num_of_Logins) as Num_of_Logins
from ComputerLabFactLevel1
group by SemYearID, TimeID, DegreeCode;
```

In short, the propagation of the new semester rules from Level-1 to Level-2 is more at the schema design level rather than at the technical implementation level.

18.3.2 Changes Not Affecting the Next Levels

There are cases where the changes to a star schema on one level will not affect the next levels. Hence, the changes are isolated to that particular star schema.

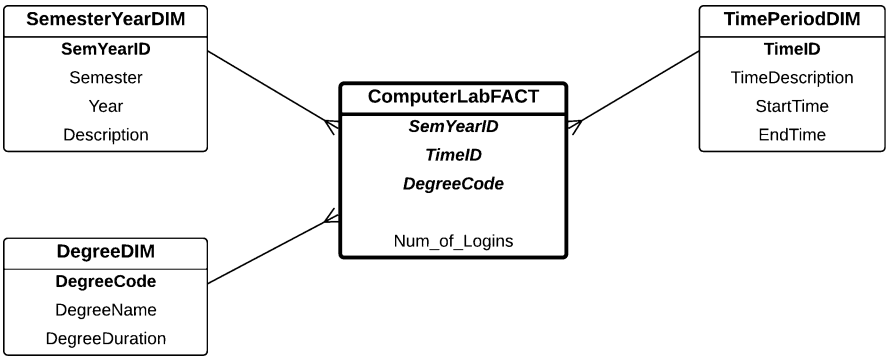

Fig. 18.9 Level-2 star schema with the Semester Year Dimension

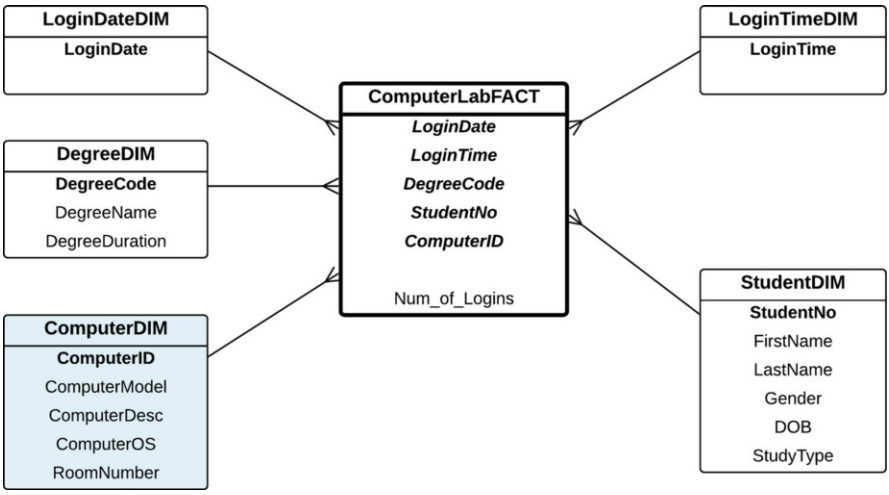

Fig. 18.10 Star schema with Computer Dimension

In the E/R diagram shown in Fig. 18.2, there is a Computer entity which is not used in the star schema. Assuming now that the requirement needs to include the Computer information in the data warehouse, the Computer Dimension needs to be added to the Level-0 star schema, as shown in Fig. 18.10. The other four dimensions are unchanged, and the fact measure also remains the same.

The coding to maintain the new Computer Dimension will be done in the same way as for the other dimensions. For the fact, we will need to get the ComputerID from the Lab Activities table.

```
create table ComputerLabFactLevel0 as
select distinct
  LoginDate, LoginTime,
  S.DegreeCode, L.StudentNo, L.ComputerID,
  count(L.StudentNo) as Num_of_Logins
from Student S, LabActivities L
where S.StudentNo = L.StudentNo
group by
  LoginDate, LoginTime,
  S.DegreeCode, L.StudentNo, L.ComputerID;
```

The FK and PK-FK constraints with the dimension tables need to be done. Also, a database trigger to automatically insert records into the fact needs to be revised to accommodate the ComputerID attribute in the Fact Table.

From the schema point of view, adding the Computer Dimension to the Level-0 star schema will not affect the next levels since the Computer Dimension will not be used in the next levels. In the Level-1 star schema, granularity is reduced by focusing on semester and time period instead of the actual Login Date and Login Time. By reducing the semester and time period granularity, the fact measure is now aggregated. There is no need to include the Computer Dimension in Level-1 as the granularity of the star schema in Level-1 is already reduced.

Adding a dimension to a star schema at one level will not affect the next levels if the dimension is not used in the next levels. Hence, the changes to one level are isolated.

18.3.3 Inserting New Star Schema

The Lab Activities case study has three levels of star schema (refer to Fig. 18.3), where Level-0 contains the raw data with Login Date and Login Time Dimensions, Level-1 is simplified to the Semester Year Dimension and Time Period Dimension (the star schema is previously shown in Fig. 18.8), and Level-2 is at which the Student Dimension is removed so that the fact measure, Number of Logins, could be more aggregated.

The jump in the granularity level from Level-0 to Level-1, that is, from Login Date and Login Time to Semester Year and Time Period (e.g. morning, afternoon, night), might be seen to be wide. It is possible that we would like to insert a new star schema between these levels. Suppose the proposed star schema is at the month and hour granularity level.

We could give this new star schema a Level-1, with the implication that the next levels are shifted upward one level: the old Level-1 becomes the new Level-2, and the old Level-2 becomes the new Level-3. Alternatively, in order to avoid any confusion in the level numbering, we could keep Level-1 and Level-2 as they were, and the new star schema is numbered Sub-Level-1, indicating that it is actually below Level-1. We will use the latter method of numbering: Level-0, Sub-Level-1, Level-1 and Level-2.

The new star schema at the Month and Hour granularity (e.g. Sub-Level-1) is shown in Fig. 18.11.

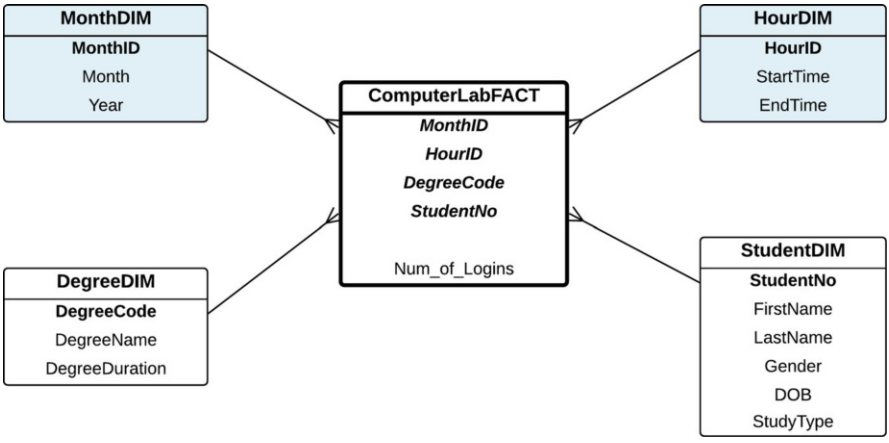

Fig. 18.11 New star schema at the Month and Hour Granularity

- **Creation of the new star schema**
  The creation of this new star schema could be done in the usual way, as described in the previous sections. Let's focus on the two new dimensions first, that is, the Month Dimension and the Hour Dimension. The initial creation of these dimension tables is as follows:

```
create table MonthDim as
select distinct
  to_char(LoginDate, 'YYYY') ||  to_char(LoginDate, 'MM')
  as MonthID,
  to_char(LoginDate, 'MON') as Month,
  to_char(LoginDate, 'YYYY') as Year
from LabActivities;

alter table MonthDim
add constraint MonthDimPK primary key (MonthID);

create table HourDim
(
  HourID            varchar2(2),
  StarTime          date,
  EndTime           date,
  constraint HourDimPK primary key  (HourID)
);

insert into HourDim
values('00', to_date('00:00', 'HH24:MI'),
to_date('00:59', 'HH24:MI'));
insert into HourDim
values('01', to_date('01:00', 'HH24:MI'),
to_date('01:59', 'HH24:MI'));
insert into HourDim
values('02', to_date('02:00', 'HH24:MI'),
to_date('02:59', 'HH24:MI'));
...
insert into HourDim
values('23', to_date('23:00', 'HH24:MI'),
to_date('23:59', 'HH24:MI'));
```

The other two dimensions, Degree and Student Dimensions, can be reused from the lower level (e.g. Level-0).

For the Fact Table, usually, the Fact Table is created using the create table command as follows.

```
create table ComputerLabFactSubLevel1 as
select distinct
  to_char(LoginDate, 'YYYY') ||  to_char(LoginDate, 'MM')
  as MonthID,
  to_char(LoginTime, 'HH24') as HourID,
  S.DegreeCode, L.StudentNo,
  count(L.StudentNo) as Num_of_Logins
```

```
from Student S, LabActivities L
where S.StudentNo = L.StudentNo
group by
  to_char(LoginDate, 'YYYY') ||  to_char(LoginDate, 'MM'),
  to_char(LoginTime, 'HH24'),
  S.DegreeCode, L.StudentNo;
```

The above method extracts the records directly from the operational database, namely, from tables Student and Lab Activities. Alternatively, we could retrieve the records from the Level-0 Fact Table, using `create view`, as follows:

```
create or replace view ComputerLabFactSubLevel1 as
select distinct
  to_char(LoginDate, 'YYYY') ||  to_char(LoginDate, 'MM')
  as MonthID,
  to_char(LoginTime, 'HH24') as HourID,
  DegreeCode, StudentNo,
  sum(StudentNo) as Num_of_Logins
from ComputerLabFactLevel0
group by
  to_char(LoginDate, 'YYYY') ||  to_char(LoginDate, 'MM'),
  to_char(LoginTime, 'HH24'),
  DegreeCode, StudentNo;
```

There are a couple of differences: (*i*) The `create view` method which uses the fact Level-0 will be simpler because the view is created and maintained automatically, whereas the `create table` method needs a separate database trigger to maintain the table for every new record inserted into the operational database. (*ii*) The `create view` method uses the `sum` function because it sums the fact measures from the Level-0 Fact Table. The `create view` method is preferable.

- **Maintaining the star schema**
  After the initial creation of this new star schema, we need to implement database triggers so that new transaction records in the operational database can immediately be incorporated into the data warehouse. The Month Dimension needs to be maintained because new transaction records may be for the new month, which is not yet in the data warehouse.

```
create or replace trigger AddMonthYear
  after insert on LabActivities
  for each row
  declare
    TempMonthID   MonthDim.MonthID%type;
    TempYear      MonthDim.Year%type;
    TempMonth     MonthDim.Month%type;
    IsFound number;
begin

  TempMonthID := to_char(:new.LoginDate, 'YYYY') ||
  to_char(:new.LoginDate, 'MM');
```

```
    TempYear := to_char(:new.LoginDate, 'YYYY');
    TempMonth := to_char(:new.LoginDate, 'MON');

    select count(*) into IsFound
    from MonthDim
    where MonthID = TempMonthID;

    if IsFound = 0 then
      insert into MonthDim
      values (TempMonthID, TempMonth, TempYear);
      commit;
    end if;

end AddMonthYear;
```

The Hour Dimension does not need to be maintained manually because the dimension contains the 24-h period when it is created. The Fact Table does not need to be maintained manually either because the Fact Table is a view rather than a table.

- **Impact on the next levels**
  The Level-1 star schema has the Semester Year Dimension and Time Period Dimension. These two dimensions were created directly using the Lab Activities table in the operational database. They do not rely on the lower-level star schemas (e.g. Level-0 and Sub-Level-1). Hence, these dimension tables will not be affected by the new Sub-Level-1 star schema based on the Month and Hour Dimensions. The TempFact was created based on the Fact Table from the Level-0 star schema. Therefore, there won't be any interdependency between the Level-1 star schema and the new Sub-Level-1 star schema. The Fact Table will not be affected either because the Level-2 star schema is based on its TempFact.

  The Level-2 star schema will not be impacted because the dimensions are reused, and the Fact Table is created using the `create view` command based on the Fact Table from the Level-1 star schema.

18.3.4 Deleting Star Schema

Following the scenario in the previous section which has four levels of star schemas, namely, Level-0, the newly inserted Sub-Level-1, Level-1 and Level-2, assume now that we want to delete the Level-1 star schema. This Level-1 star schema has the granularity of Semester Year and Time Period (e.g. morning, afternoon, night) but still maintains the Student Dimension. Note that only the top level, which in this case is Level-2, does not have the Student Dimension, making the aggregation level higher.

There are a couple of implications for the Level-2 star schema when the Level-1 star schema is deleted.

- **Dimensions**
  The Level-2 star schema shares three dimensions with the Level-1 star schema, namely, Semester Year Dimension, Time Period Dimension and Degree Dimension. When deleting the Level-1 star schema, we only remove the Fact Table; the dimension tables are not removed as they are still being used by the Level-2 star schema.

- **Fact**
  The Fact Table for Level-2 is created using the `create view` command based on the Fact Table in Level-1. Since the Level-1 Fact Table is now removed, this will have an impact on the Fact Table for Level-2. Actually, the semantics of the Level-2 and Level-1 star schema in this case study is very similar. The Level-1 star schema has four dimensions, that is, Semester Year, Time Period, Degree and Student Dimensions, whereas the Level-2 star schema has only the first three dimensions, without the Student Dimension. The fact measure is also the same, which is an aggregate value of Number of Logins.

  The creation of the Semester Year Dimension and Time Period Dimension has been discussed at length in the previous section. This was then followed by the creation of the TempFact Table and then the final Fact Table. In the TempFact Table, a temporary Fact Table is created to which two new attributes are added: one attribute to accommodate the Semester Year information and the other attribute for the Time Period information. Once the TempFact is created, the final Fact Table aggregates the records from the TempFact Table based on the four dimension keys: SemYearID, TimeID, Degree Code and StudentNo.

  We could reuse this method to create the new Fact Table for the Level-2 star schema as the Fact Table for the Level-1 star schema is to be removed. The Fact Table could reuse the TempFact, but when it is aggregated, it does not need to include StudentNo as the Student Dimension is not used at this level.

```
create table ComputerLabFactLevelNew2 as
select
  SemYearID, TimeID, DegreeCode,
  count(StudentNo) as Num_of_Logins
from TempComputerLabFactLevel1
group by SemYearID, TimeID, DegreeCode;
```

  The database trigger to maintain the Fact Table could also reuse the same database trigger to update Fact Level-1, as previously described.

```
create or replace trigger UpdateFactLevel2
  after insert on TempComputerLabFactLevel1
  for each row
  declare
    IsFound number;
begin
  select count(*) into IsFound
  from ComputerLabFactLevelNew2
  where  SemYearID = :new.SemYearID
    and TimeID = :new.TimeID
    and DegreeCode = :new.DegreeCode;
```

```
    if IsFound <> 0 then
      update ComputerLabFactLevelNew2
      set Num_of_Logins = Num_of_Logins + 1
      where SemYearID = :new.SemYearID
        and TimeID = :new.TimeID
        and DegreeCode = :new.DegreeCode;
    else
      insert into ComputerLabFactLevelNew2
      values (:new.SemYearID, :new.TimeID,
        :new.DegreeCode, 1);
     end if;
     commit;
  end UpdateFactLevel2;
```

18.4 Operational Database Evolution

The operational databases which feed into the data warehouse may also evolve over time. This may then impact the data warehouse, as in Active Data Warehousing, updates or changes in the operational databases are reflected in the data warehouse in real time.

As the operational database is changed, the data warehousing requirements may also change. In this section, we are going to see how the changes in the operational database (at various degrees of severity) will change the data warehousing requirements from small changes to the table structure and changes to the entities and relationships to major changes to the operational database.

Some of the changes in the operational database may affect the Level-0 star schema only, while some will propagate to all star schemas in various levels of granularity. In the worst scenario, the entire data warehouse (e.g. all star schemas) will be overhauled, and the old data warehouse is fully decommissioned and replaced with the new one.

18.4.1 Changes in the Table Structure

As the operational or transactional system is used over the time, additional information which was not captured in the past is not being captured, and this additional information may affect the data warehouse. Using the Computer Lab Activities case study, in the original system, the Login Time is recorded but not the Logout Time. Supposing the Logout Time is recorded in the Lab Activities table in the operational database, this information (e.g. Logout Time) or the derived information (e.g. Duration of Login) can be included in the data warehouse. This will impact not only the dimension of the star schema but also the fact measure. So,

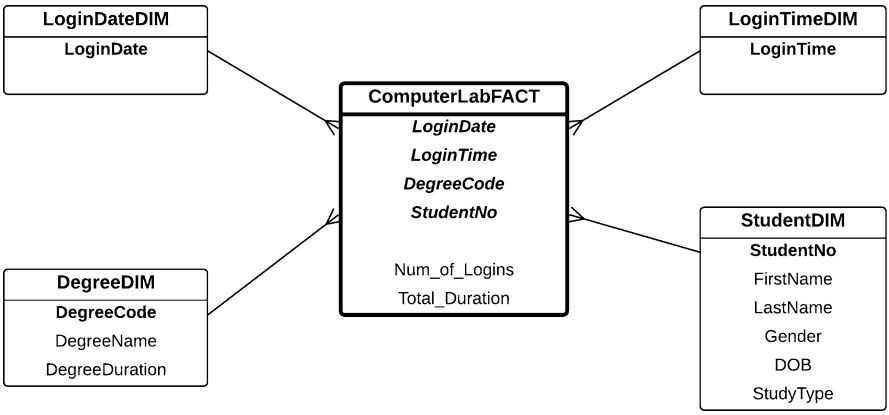

Fig. 18.12 Star schema with Total Duration fact measure

this is an example of how the table structure in the operational database, which in this case is the Lab Activities table structure, is changed by adding new information such as Logout Time.

By having Logout Time information in the operational database, we can measure the duration of the login activity added to the data warehouse. In this case, a new fact measure, Duration (or Total Duration), can be added to the star schema. Figure 18.12 shows a Level-0 star schema, where the new fact measure, Total Duration, has been added to the fact. As a comparison, the original star schema is shown in Fig. 18.3.

Because there is a new fact measure in the Level-0 star schema, the fact measure will likely be carried forward to all upper-level star schemas. This implies that the changes in the table structure in the operational database, in this case the Lab Activities table, affect the Level-0 star schema, which subsequently has an impact on all the upper-level star schemas.

There are two possible options to deal with this. Option 1 is to update the existing star schemas. In this case, the existing Level-0 star schema is updated to include a new fact measure, namely, Total Duration in the Fact Table. All existing records in the Fact Table will have a null value in the Total Duration fact measure because the old records do not have this information. But for every new record added to the Fact Table, Total Duration will be recorded. Technically, two actions need to be taken: Firstly, the Fact Table structure needs to be altered by adding a new column for this new fact measure, and this attribute needs to be filled with a null value in each record. Secondly, to correctly insert the new records into the Fact Table, the database trigger needs to be revised so that the Total Duration information can be inserted correctly for every new record in the Fact Table. This approach needs to be taken in each level of star schema in the data warehouse.

Option 2 is to build a new data warehouse, since the star schema now has new requirements. The old data warehouse, which does not have the new information on Total Duration, will be decommissioned from an Active Data Warehousing. The

old data warehouse will still be used since all the historical data is captured there. However, the new data will be kept in the new data warehouse.

Since the second option is to build a new data warehouse, the steps that need to be taken are the same as when building a new data warehouse, which has already been discussed in this book. Therefore, in this section, the first option, which is altering and updating the existing star schemas, will be discussed.

The existing Level-0 Fact Table needs to be altered by adding a new attribute, which is then set as a null value.

```
alter table ComputerLabFactLevel0
add Total_Duration date;
```

To handle the new records, the following database trigger can be used. The Total Duration is the difference between Logout Time and Login Time.

```
create or replace trigger UpdateFactLevelNew0
  after insert on LabActivities
  for each row
  declare
    Degree Student.DegreeCode%type;
begin
  select DegreeCode into Degree
  from Student
  where StudentNo = :new.StudentNo;

  insert into ComputerLabFactLevel0
  values (:new.LoginDate, :new.LoginTime,
    :Degree, :new.StudentNo, 1,
    :new.LogoutTime - :new.LoginTime);
  commit;
end UpdateFactLevelNew0;
```

Changes in the upper levels can be done in a similar way. Figure 18.13 shows an example of the Level-1 star schema where the Total Duration fact measure is added to the fact.

The implementation method to update this star schema is to alter the Fact Table and change the database trigger.

```
alter table ComputerLabFactLevel1
add Total_Duration date;

create or replace trigger UpdateFactLevel1
after insert on TempComputerLabFactLevel1
  for each row
  declare
    IsFound number;
begin
  select count(*) into IsFound
  from ComputerLabFactLevel1
  where  SemYearID = :new.SemYearID
    and TimeID = :new.TimeID
    and DegreeCode = :new.DegreeCode
    and StudentNo = :new.StudentNo;
```

```
    if IsFound <> 0 then
      update ComputerLabFactLevel1
      set Num_of_Logins = Num_of_Logins + 1,
      Total_Duration = Total_Duration + :new.Total_Duration
      where SemYearID = :new.SemYearID
        and TimeID = :new.TimeID
        and DegreeCode = :new.DegreeCode
        and StudentNo = :new.StudentNo;
    else
      insert ComputerLabFactLevel1
      values (:new.SemYearID, :new.TimeID,
        :new.DegreeCode, :new.StudentNo, 1,
        :new.Total_Duration);
    end if;
    commit;
  end UpdateFactLevel1;
```

The database trigger checks if the new record inserted into the TempFact is a record with a new Semester Year, which does not yet exist in the Fact Table, and then it will insert the new record with Total Duration, which is the same as the Total Duration in the TempFact. On the other hand, if the record is a record with an existing Semester Year, the trigger will update the fact record and increment the Total Duration by the Duration of the new record in the TempFact.

As the fact is based on the TempFact, the database trigger for the TempFact also needs to be adjusted, as follows. After the TempFact Table is altered, the database trigger inserts the record into the TempFact Table, where Total Duration is the difference between Logout Time and Login Time.

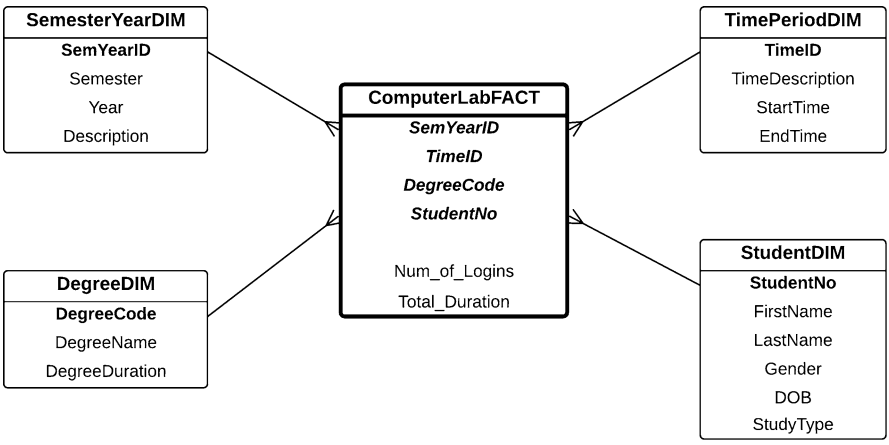

Fig. 18.13 Level-1 star schema with Total Duration fact measure

```
alter table TempComputerLabFactLevel1
add Total_Duration date;

create or replace trigger UpdateTempFactLevel1
  after insert on ComputerLabFactLevel0
  for each row
  declare
    TempSemYearID TempComputerFactLabLevel1.SemYearID%type;
    TempTimeID    TempComputerFactLabLevel1.TimeID%type;
begin
  -- the codes for checking the Semester Year information
  -- based on Login Date are here
  -- for simplicity, the codes are not included here

  -- the codes for checking the Time Period information
  -- based on Login Time are here
  -- for simplicity, the codes are not included here

  insert into TempComputerLabFactLevel1
  values (:new.LoginDate, :new.LoginTime,
  :new.DegreeCode, :new.StudentNo,
  TempSemYearID, TempTimeID,
  :new.LogoutTime - :new.LoginTime);
  commit;
end UpdateTempFactLevel1;
```

18.4.2 Changes in the E/R Schema

Another type of change in the operational database is the addition of new entities (and hence new relationships) in the E/R diagram. This is simply due to the additional information being available and recorded in the new operational system. The original E/R diagram, which is used in the Computer Lab case study, is shown in Fig. 18.2.

Two areas of extension are applied in this case study. The first extension is to create an entity to store the Room information. In the current E/R diagram, the Room information is merely stored as an attribute in the Computer entity. The Room information is now expanded to a new entity, called the Room entity, with attributes such as Room Number, Building, Number, etc. With more information about Rooms, it is clear that the decision-makers might want to analyse the lab usage of each room, for example. Therefore, the data warehouse will subsequently need to change.

The second extension relates to the activities in the lab. At the moment, each student login is recorded with information including Login Date, Login Time (and Logout Time), ComputerID and, of course, StudentNo. With the new audit trial, each login activity will include the software and tools accessed in each login connection as well. This information was not available in the original E/R diagram but will now be included. During one lab attendance, any software accessed by

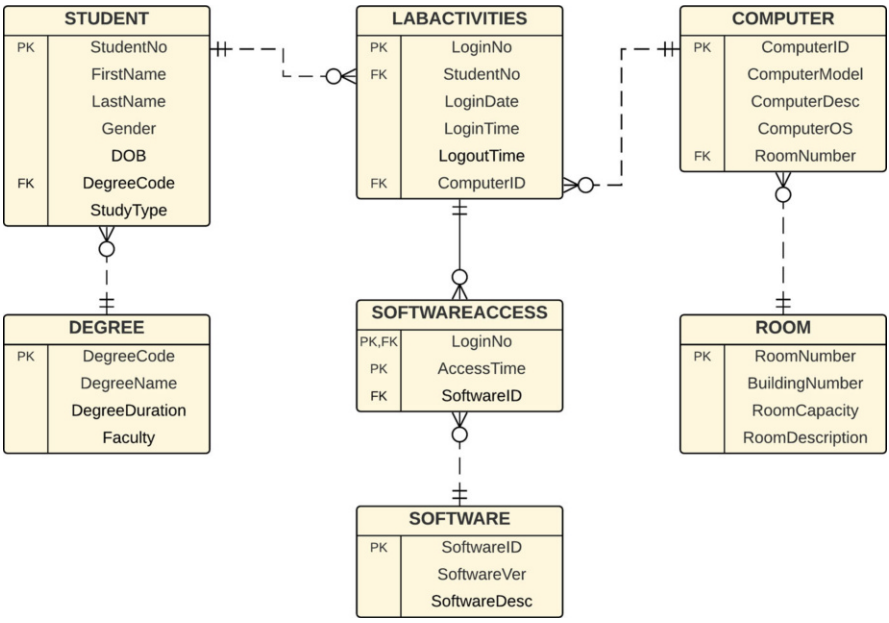

Fig. 18.14 Revised E/R diagram

the student will be recorded. This results in an extension of two entities: Software Access and Software entities.

These two extensions involve the creation of new entities and relationships (as opposed to internal changes within one entity as described in the section above). The new E/R diagram is shown in Fig. 18.14.

The data warehouse is now expanded into two Level-0s. One is at the login granularity and the second is at the software access granularity. Note that for each login, a student may access several software applications. Clearly, these are two levels of granularity which cannot be combined into one star schema.

The Level-0a star schema as shown in Fig. 18.15 focuses on the Computer Room granularity. The fact measures are Number of Logins and Total Duration. Number of Logins basically counts the number of records in the Lab Activity table, as one record in the Lab Activity table is one login activity.

The Level-0b star schema as shown in Fig. 18.16 focuses on the Software Access granularity. The original fact measure Number of Logins is not applicable because Number of Logins is counted per student computer login. In this star schema, one computer login may access several software applications. Therefore, the fact measure must count the Total Access to any software rather than each computer login. The second difference is that there is no Total Duration because there is no information about the duration the student accessed the software at each computer login.

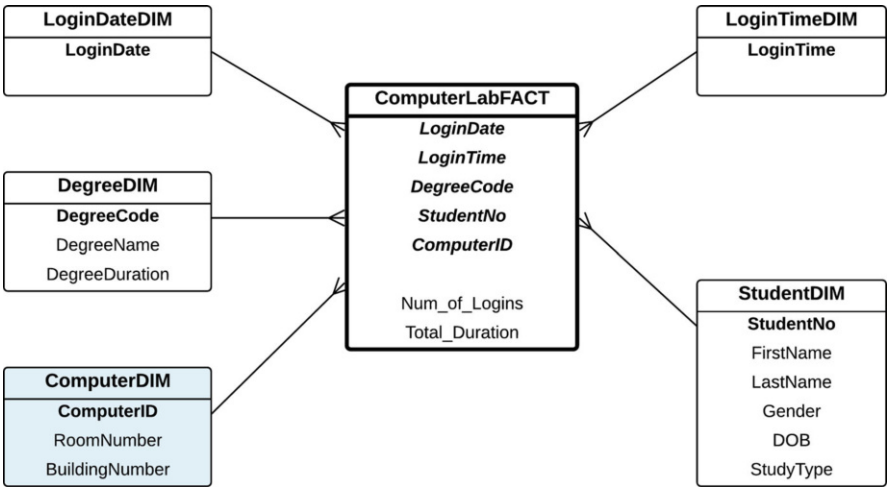

Fig. 18.15 Level-0a star schema at a Computer Room granularity level

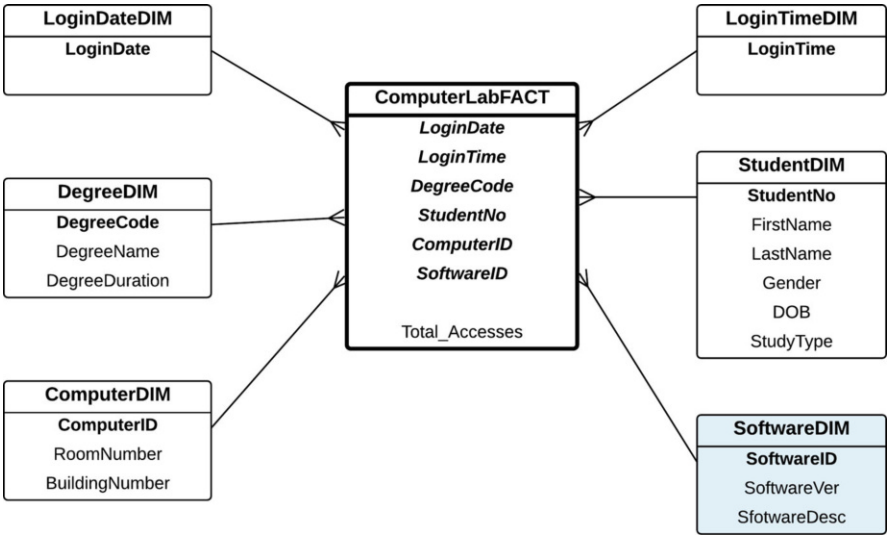

Fig. 18.16 Level-0b star schema at a Software Access granularity level

There are only Login Time and Logout Time but are at the computer login level, not at a specific software level. Therefore, the Total Duration fact measure is not applicable here.

Because the Level-0a star schema is an extension of the original star schema, we can simply reuse the old star schema by adding it to the Computer Dimension. Then, the database trigger needs to also insert ComputerID into the Fact Table.

```
alter table ComputerLabFactLevelNew0a
add ComputerID varchar2(20);

create or replace trigger UpdateFactLevelNew0a
  after insert on LabActivities
  for each row
  declare
    Degree Student.DegreeCode%type;
begin
  select DegreeCode into Degree
  from Student
  where StudentNo = :new.StudentNo;

  insert into ComputerFactLabLevelNew0a
  values (:new.LoginDate, :new.LoginTime,
    Degree, :new.StudentNo, 1,
    :new.LogoutTime - :new.LoginTime,
    :new.ComputerID);
  commit;
end UpdateFactLevelNew0a;
```

The Computer Dimension can be created in the usual way, either using `create table` followed by a database trigger or `create view`. The following illustrates the `create view` method:

```
create or replace view ComputerDim as
select distinct
  C.ComputerID, C.RoomNumber, R.BuildingNumber
from Computer C, Room R
where C.RoomNumber = R.RoomNumber;
```

The Level-0b star schema is a new star schema that needs to be created. The Fact Table basically includes a join operation with the Software Access table, and the `count` aggregate function counts the join results because the granularity of the fact measure is at each software access level. The Software Access Dimension can be created the same way as the other dimensions.

```
create table ComputerLabFactLevelNew0b as
select distinct
  LoginDate, LoginTime,
  S.DegreeCode, L.StudentNo,
  A.SoftwareID,
  count(*) as Total_Accesses
from Student S, LabActivities L, SoftwareAccess A
where S.StudentNo = L.StudentNo
and L.LoginNo = A.LoginNo
group by
  LoginDate, LoginTime,
  S.DegreeCode, L.StudentNo, A.SoftwareID;
```

The Fact Table needs a database trigger to trigger an automatic update to the Fact Table for every new transaction record added in the operational database. Note that the database trigger is based on an insertion into the Software Access table. For

each new record inserted into the Software Access table in the operational database, a new record will be inserted into the Fact Table.

```
create or replace trigger UpdateFactLevelNew0b
  after insert on SoftwareAccess
  for each row
  declare
    StudentNo    LabActivities.StudentNo%type;
    LoginDate    LabActivities.LoginDate%type;
    LoginTime    LabActivities.LoginTime%type;
    ComputerID   LabActivities.ComputerID%type;
    Degree       Student.DegreeCode%type;
begin
  select StudentNo, LoginDate, LoginTime, ComputerID
  into StudentNo, LoginDate, LoginTime, ComputerID
  from LabActivities
  where LoginNo = :new.LoginNo;

  select DegreeCode into Degree
  from Student
  where StudentNo = :StudentNo;

  insert into ComputerLabFactLevelNew0b
  values (LoginDate, LoginTime,
    Degree, StudentNo, ComputerID,
    :new.SoftwareID, 1);

  commit;
end UpdateFactLevelNew0b;
```

Changes to Level-0 star schemas will be propagated to the next levels. The method discussed previously relating to propagating changes to the next levels can be used.

18.4.3 Changes in the Operational Database

All the operational databases, in which the data warehouse is based, may change. The old system is decommissioned and the new system is deployed. Table names may change; table structures may change; the entire design may change. In other words, the old operational database evolves into a new one. These are the most complex changes related to the operational databases, and we need to look at the impact it has on Active Data Warehousing.

The E/R diagram for the operational database that keeps track of the computer lab activities is shown previously in Fig. 18.14, where the Degree entity is included in this schema. Suppose now the system that stores the computer lab activities is changed, and then the Degree entity is no longer there (see Fig. 18.17 for the new E/R diagram for the Computer Lab Activities).

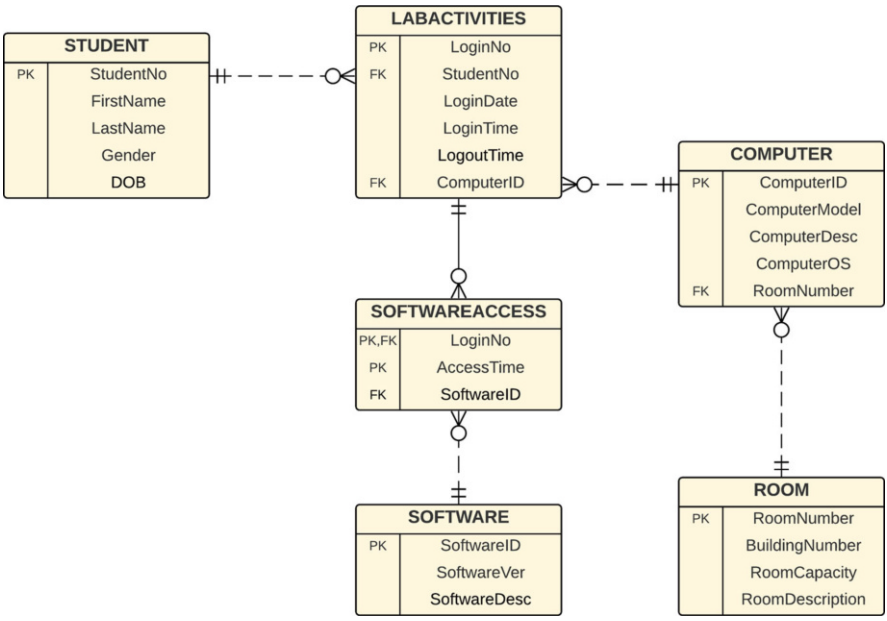

Fig. 18.17 New Computer Lab Activities E/R diagram

There is also a new system that keeps the enrolment records of students, which is shown in the E/R diagram in Fig. 18.18. It keeps track of every individual subject in which each student enrols. It also calculates the WAM (weighted average mark) and GPA (grade point average) of each degree the student has completed at this university. Note that a student may have completed several degrees at this university.

Complexity arises because the data warehouse for the Computer Lab Activities has the Degree Dimension and the Degree entity is no longer in the Computer Lab Activities E/R diagram but now exists in the Student Enrolment E/R diagram. These are two separate operational databases which may run on different systems.

The star schema, as shown in Fig. 18.15, is a Level-0 star schema with five dimensions: Login Date, Login Time, Student, Degree and Computer Dimensions. There are two options to deal with Active Data Warehousing when the operational databases have changed.

- **Option 1: Update existing star schema**
  Option 1 is to update the structure (and data) of the star schema, whenever possible, which in this case is the star schema in Fig. 18.15. Four dimensions, except the Degree Dimensions, are available from the new Computer Lab Activities database. The Degree information needs to be taken from the new Student Enrolment database. Hence, in the database trigger for the fact, it needs to obtain the Degree information from the Student Enrolment system, assuming that the Student Enrolment system is accessible by the database trigger.

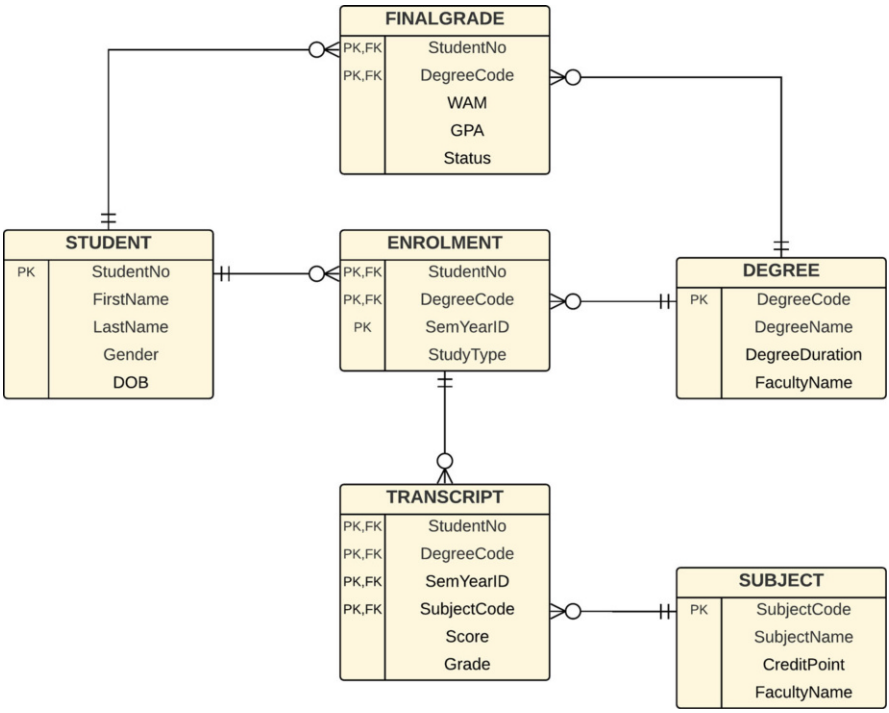

Fig. 18.18 New Student Enrolment E/R diagram

```
create or replace trigger UpdateFactLevelNew0a
  after insert on LabActivities
  for each row
  declare
    Degree
      StudentEnrolmentDatabase.Enrolment.DegreeCode%type;
    TempSem varchar2(2);
    TempSemYearID
      StudentEnrolmentDatabase.Enrolment.SemYearID%type;
begin
  if (to_char(:new.LoginDate, 'MMDD') >= '0101')
  and (to_char(:new.LoginDate, 'MMDD') <= '0630') then
    TempSem = 'S1'
  else
    TempSem = 'S2'
  end if;
  TempSemYearID = to_char(:new.LoginDate, 'YYYY') ||
    TempSem;

  select distinct E.DegreeCode into Degree
  from StudentEnrolmentDatabase.Enrolment E
  where E.StudentNo = :new.StudentNo
  and SemYearID = TempSemYearID;
```

```
    insert into ComputerFactLabLevelNew0a
    values (:new.LoginDate, :new.LoginTime,
      Degree, :new.StudentNo, 1,
      :new.LogoutTime - :new.LoginTime,
      :new.ComputerID);
    commit;
  end UpdateFactLevelNew0a;
```

The Degree Dimension must also be altered so it will obtain the new degree records from the Student Enrolment system, particularly from the Degree table.

- **Option 2: Create new star schema**
  Option 2 is to create new star schemas in all the required granularity levels in the data warehouse. The old star schemas are then decommissioned from the Active Data Warehousing. The old star schemas can still be used as a Passive Data Warehousing to analyse the historical data.

  Because there are two separate operational databases, firstly, we need to create a separate star schema from each operational database, and after that, the two star schemas are combined. Creating two separate star schemas from their respective operational databases is a suitable approach, particularly when the two operational databases are in two different systems, and it may not be possible to access both of them from one system.

  The first star schema is based on the Computer Lab Activities operational database and is shown in Fig. 18.19. This star schema looks very close to the desired final star schema in Fig. 18.15. The only difference is that in the star schema in Fig. 18.19, there is no Degree Dimension. This is simply because the Degree information is no longer available in the operational database of the Computer Lab Activities (refer to Fig. 18.17).

  The second star schema is based on the Student Enrolment operational database as shown in Fig. 18.20. It includes the Degree Dimension, which is

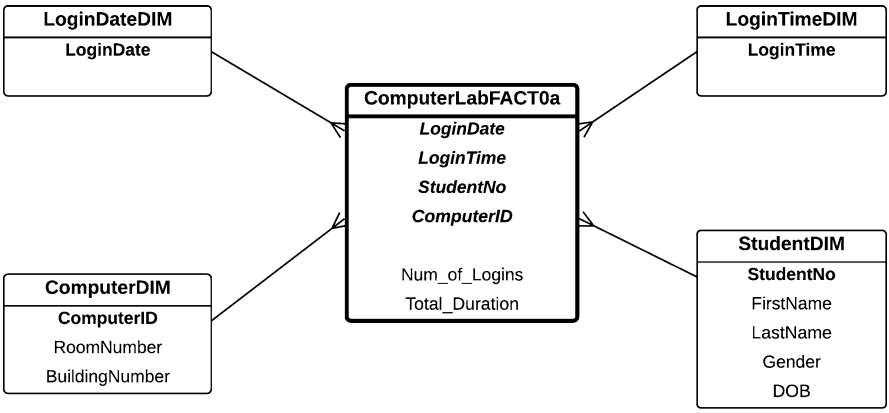

Fig. 18.19 New Computer Lab Activities star schema

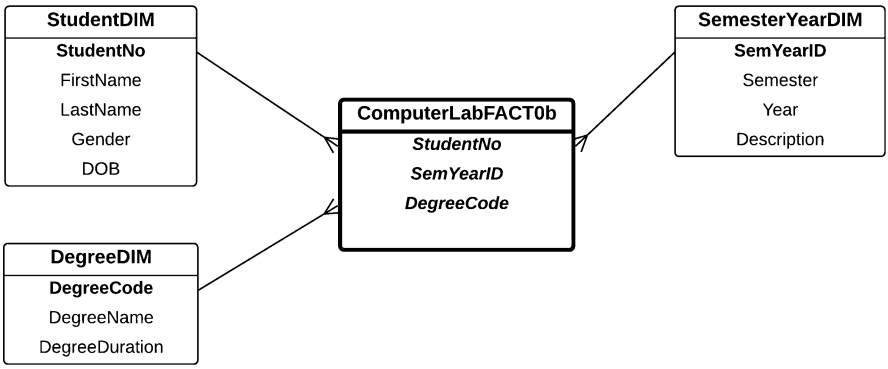

Fig. 18.20 New Student Enrolment star schema

needed in the final star schema, as well as the Student and Semester Year Dimension.

The difference between the first and second star schemas is not only in the Degree Dimension. The granularity seems to be different too as the second star schema uses the Semester Year Dimension, not the Login Date Dimension, as in the first star schema. The second star schema needs to use the Semester Year Dimension because the Student Enrolment system records the semester year information; hence, the date information is irrelevant. Another difference between the first and second star schemas is that in the second star schema, there is no fact measure. The reason for this is that the second star schema will be used as a "lookup table" to obtain the respective Degree information for each student in the first star schema. So, the second star schema is not meant to be an independent star schema.

Both star schemas can be created in the usual way as discussed previously. Once they are created, they need to be merged to form the final star schema. The first step is to copy the first star schema (e.g. ComputerLabFact0a) to the final star schema (e.g. ComputerLabFactNew0a) and add a new column for the Degree Code.

```
create table ComputerLabFactNew0a as
select * from ComputerLabFact0a;

alter table ComputerLabFactNew0a
add (
  DegreeCode varchar2(10),
  SemYearID     varchar2(10),
  Semester      varchar2(2),
  Year          varchar2(4));
```

Because there is a different granularity level between the two star schemas, we need to have a temporary attribute to store the semester year information in the

final Fact Table. Then, the SemYearID column is filled with the corresponding semester year information based on the Login Date.

```
update ComputerLabFactNew0a
set Semester = 'S1'
where to_char(LoginDate, 'MM/DD') >= '01/01'
and to_char(LoginDate, 'MM/DD') <= '06/30';

update ComputerLabFactNew0a
set Semester = 'S2'
where Semester is null;

update ComputerLabFactNew0a
set Year = to_char(LoginDate, 'YYYY');

update ComputerLabFactNew0a
set SemYearID = Year || Semester;

alter table ComputerLabFactNew0a
drop (Semester, Year);
```

Finally, the final star schema is updated by filling in the correct Degree Code for each student, where the Semester Year information matches the two star schemas.

```
update ComputerLabFactNew0a
set DegreeCode = ComputerLabFact0b.DegreeCode
where StudentNo = ComputerLabFact0b.StudentNo
and SemYearID = ComputerLabFact0b.SemYearID;

alter table ComputerLabFactNew0a
drop SemYearID;
```

Once the final Fact Table is created, we need a database trigger that will automatically insert records into the final Fact Table when there are new records in the first Fact Table (e.g. ComputerLabFact0a). The trigger needs to convert the Login Date into the equivalent SemYearID, and then based on this SemYearID (as well as the StudentNo), it can search for the correct Degree Code in the second Fact Table (e.g. ComputerLabFact0b). An insertion into the final Fact Table can then be executed.

In short, the second Fact Table (e.g. ComputerLabFact0b) is only used as a lookup table to obtain the Degree Code for each student, based on the Login Date information.

```
create or replace trigger UpdateComputerLabNewFactLevel0a
  after insert on ComputerLabFact0a
  for each row
  declare
    TempSemesterID  SemesterYearDim.Semester%type;
    TempSemYearID   ComputerLabFact0b.SemYearID%type;
    TempDegree      ComputerLabFact0b.DegreeCode%type;
```

```
begin
  if (to_char(:new.LoginDate, 'MM/DD') >= '01/01')
    and (to_char(:new.LoginDate, 'MM/DD') <= '06/30') then
    TempSemesterID := 'S1';
  else
    TempSemesterID := 'S2';
  end if;

  TempSemYearID := to_char(:new.LoginDate, 'YYYY') ||
    TempSemesterID;

  select DegreeCode into TempDegree
  from ComputerLabFact0b
  where StudentNo = :new.StudentNo
  and SemYearID = TempSemYearID;

  insert into ComputerLabFactNew0a
  values (:new.LoginTime, :new.LoginDate,
    TempDegree, :new.StudentNo, :new.ComputerID,
    :new.Num_of_Logins, :new.Total_Duration);
  commit;
end UpdateComputerLabNewFactLevel0a;
```

When there are changes in the Level-0 star schema, these changes may propagate to the upper levels.

18.5 Summary

Active Data Warehousing is solely about interdependency between the operational database, which is the source of the data warehouse and the data warehouse itself, as well as among the star schemas of various levels of granularity in the data warehouse. This interdependency can be very complex and can be complicated to maintain.

There are three types of interdependencies studied in this chapter: (i) incremental updates, which is when new records are inserted into or old records are deleted from the operational database, they are immediately reflected in the star schema; (ii) changes in the data warehouse, which is when the data warehouse creates a new star schema or deletes an existing star schema, as well as changes to the upper levels of star schemas when the lower-level star schemas change; and (iii) changes in the operational database, which is when the underlying operational database has changed and has an impact on the data warehouse.

Since interdependency can be quite complex in many cases, Active Data Warehousing is used only for applications that really need active data in the data warehouse. This includes the monitoring system which may rely on real-time data and not so much on historical data. In this case, an Active Data Warehousing is appropriate and useful.

18.6 Exercises

18.1 Using the case study discussed in the "Automatic Updates of Data Warehouse" section earlier in this chapter, write the complete SQL commands for Level-0, Level-1 and Level-2 star schemas. Test the system by inserting new records into the operational database, and check that the correct updates have been made in all levels of the star schemas in the data warehouse.

18.2 Using the case study discussed in the "Changes Propagating to the Next Levels" section in this chapter, write the complete SQL commands for Level-0, Level-1 and Level-2 star schemas, where the Semester Year Dimension is added to the Level-1 star schema. Test the system by adding new transaction records with new Semester Year information, which is not in the previous transaction records, and check if the Semester Year Dimension is correctly implemented, as well as in the fact of each level of the star schemas.

18.3 Using the case study discussed in Sect. 18.4.1, write the complete SQL commands for Level-0 and Level-1 star schemas, where the Total Duration fact measure is added to the star schemas. Test the system by adding new transaction records with Logout Time information, and check if the Total Duration fact measure is calculated correctly in both levels of the star schemas.

18.7 Further Readings

Active Data Warehousing is about schema evolution, that is, how schema evolution affects the data warehouse. The schema evolution can either involve the databases or the data warehouses. [1] discusses the effects of evolution on the database design. Other database design books, such as [2–5] and [6], described the concept of evolution in database design.

Active Data Warehousing uses triggers and views to maintain the effect of changes. Further details on these SQL commands can be found in various SQL books, including [7–10].

Many of the well-known database textbooks explain the use of database triggers and views using SQL [11–24].

References

1. S.W. Ambler, P.J. Sadalage, *Refactoring Databases: Evolutionary Database Design*. Addison-Wesley Signature Series (Fowler) (Pearson Education, London, 2006)
2. S. Bagui, R. Earp, Database design using entity-relationship diagrams, in *Foundations of Database Design* (CRC Press, New York, 2003)