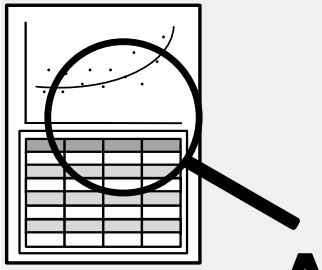


Active Data Warehousing

Outline

- An **Active Data Warehousing** is where the data warehouse is immediately updated when the operational database is updated.
- This chapter discusses the complexities in three parts, which are:
  1) Incremental updates
  2) Data warehousing schema evolution
  3) Operational database evolution

1 Passive vs. Active Data Warehousing

- A **Passive Data Warehousing** is a data warehouse that, once built, will remain unchanged. 一成不變

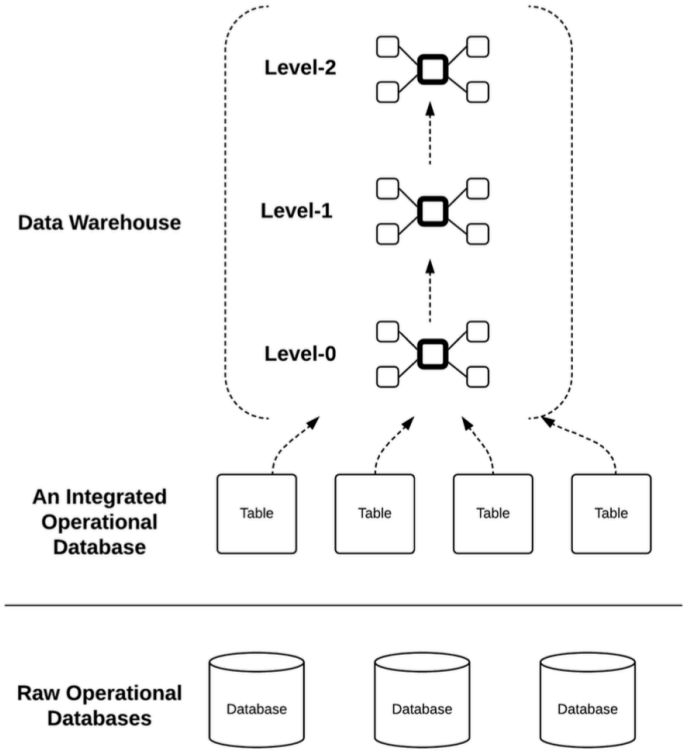
- An **Active Data Warehousing** is dynamic. 一成可變

- When the operational database is updated, the data warehouse is immediately updated. The data warehouse is always up-to-date, and it is no longer purely historical.

1 Passive vs. Active Data Warehousing

- The architecture of Active Data Warehousing

Bottom up



1 Passive vs. Active Data Warehousing

- 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 is often easier to 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).

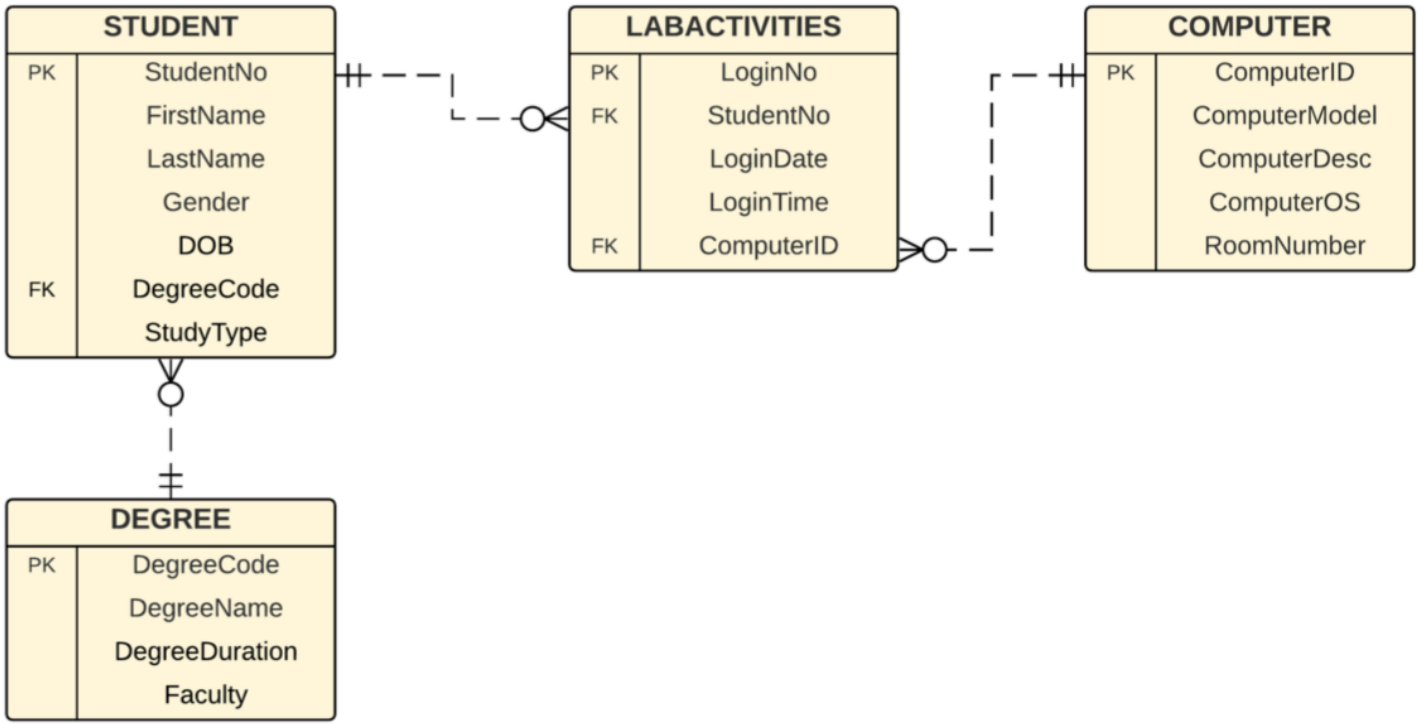
2 Incremental Updates

- **Incremental updates** in Active Data Warehousing are not to automatically update the data warehouse.
- Other elements need to be considered, 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.
- For example: data has an expire date; rules changed

2.1 Automatic Updates of Data Warehouse

- **Automatic updates** refer to updates that occur immediately once the operational database has changed.
- In the Computer Lab Activities case study, It is a simple database that consists of four tables: Lab Activities, Computer, Student, and Degree.

2.1 Automatic Updates of Data Warehouse



2.1 Automatic Updates of Data Warehouse

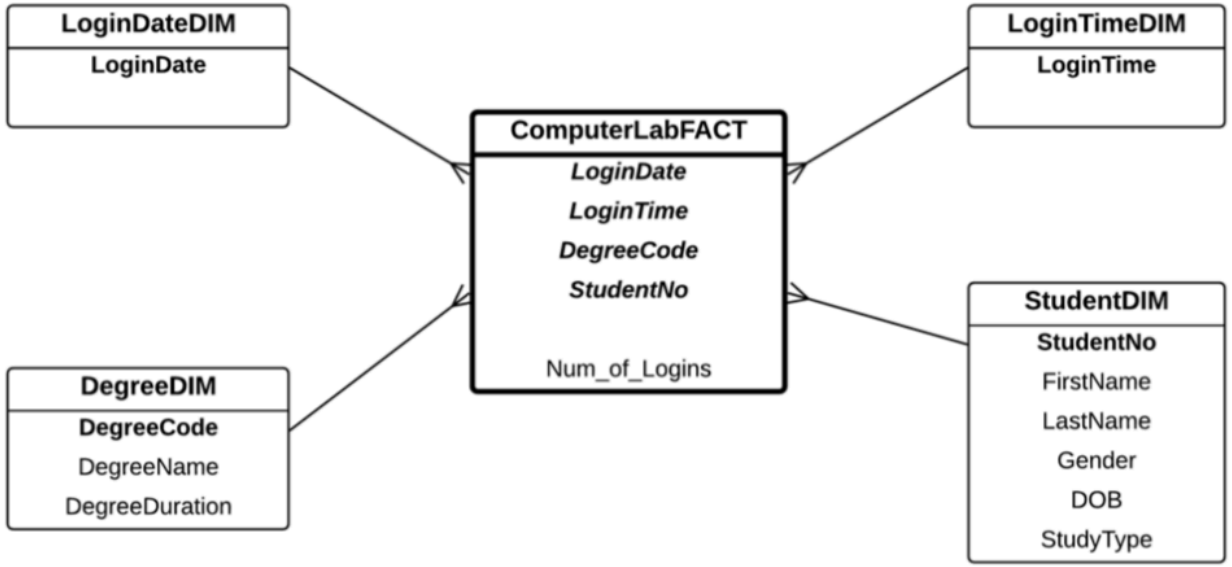
- The data warehouse focuses on one fact measure: Number_of_Usage (or Number_of_Logins).

- The data warehouse consists of three levels of granularity.

- The Level-0 star schema (or the lowest level) does not have any aggregation in the fact measure.
- The level of aggregation increases as the level of the star schema increases in Level-1 and Level-2 star schemas.

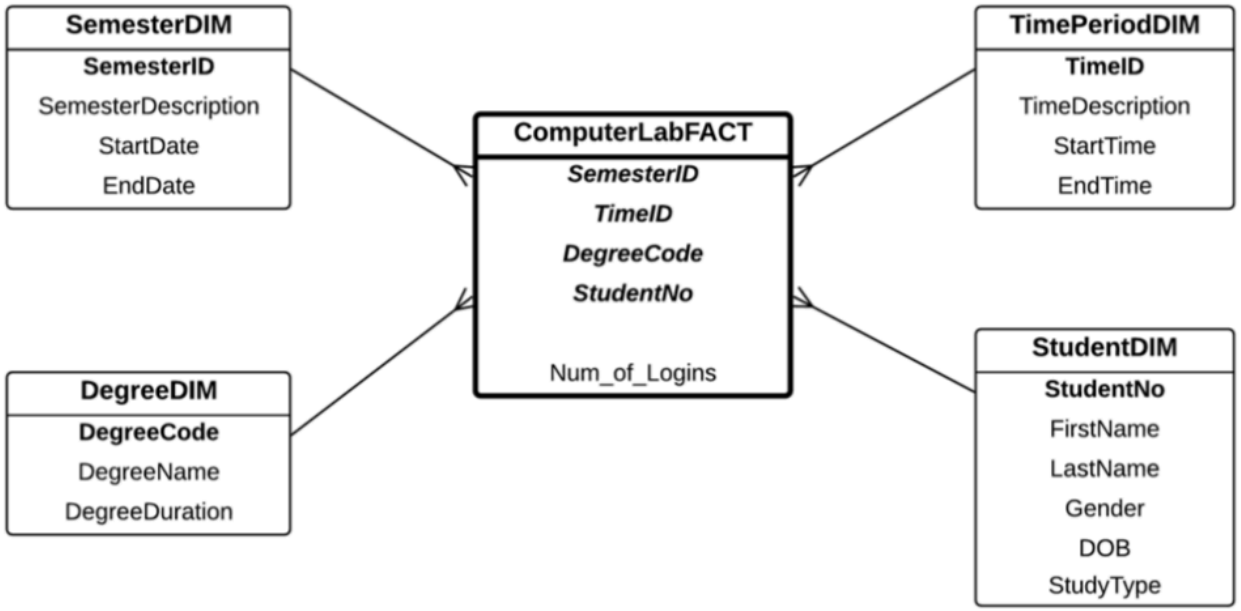
2.1 Automatic Updates of Data Warehouse

- Level-0 Star Schema



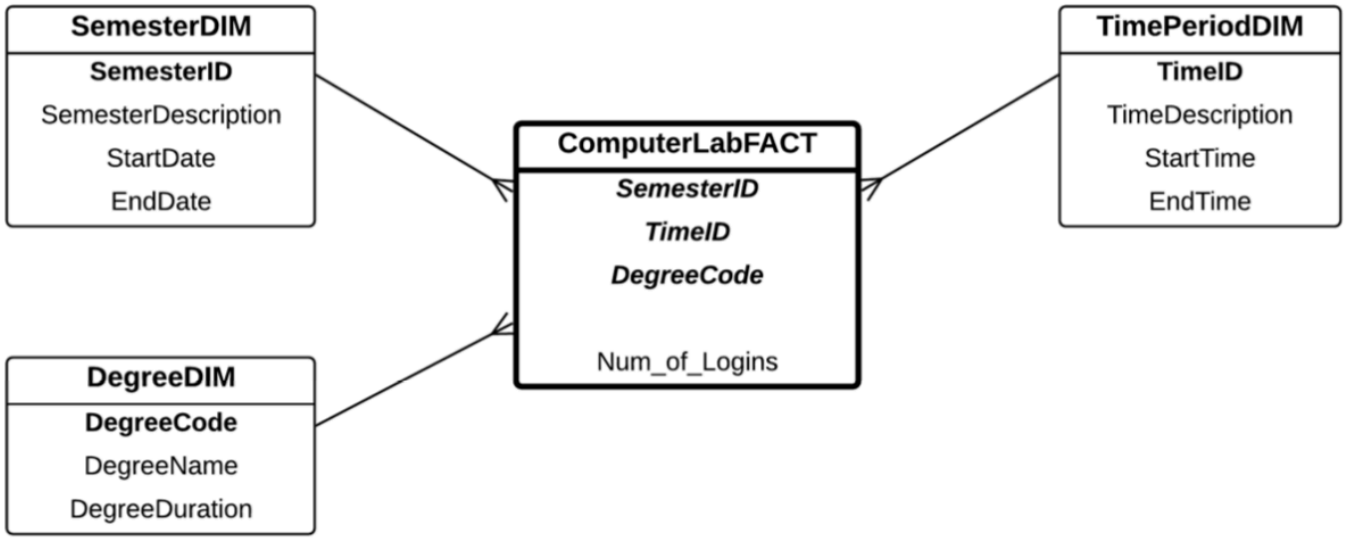
2.1 Automatic Updates of Data Warehouse

- Level-1 Star Schema



2.1 Automatic Updates of Data Warehouse

- Level-2 Star Schema



2.1 Automatic Updates of Data Warehouse - Level 0

- The first step in star schema implementation is to create dimension tables, and then followed by the TempFact and Fact Tables.
- Assuming that the operational database that contains the tables are 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.

2.1 Automatic Updates of Data Warehouse - Level 0

```
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;
```

2.1 Automatic Updates of Data Warehouse - Level 0

- 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.

2.1 Automatic Updates of Data Warehouse - Level 0

- Take the Student Dimension as an example:

```
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;
```

2.1 Automatic Updates of Data Warehouse - Level 0

- 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.

2.1 Automatic Updates of Data Warehouse - Level 0

- Use ***create table*** and then use a database trigger to automatically add a record to the fact when there is a new record is inserted into the transaction table

2.1 Automatic Updates of Data Warehouse - Level 0

```
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;
```

2.1 Automatic Updates of Data Warehouse - Level 0

- 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;
```

2.1 Automatic Updates of Data Warehouse - Level 0

- **PK-FK Constraints**

- In Passive Data Warehousing, there is no need to create a Primary Key - Foreign Key constraint between each dimension key with the fact, because once the star schema is created and built, no records will be updated in the star schema there will be no insert as the data warehouse is passive and no delete.
- 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 minimize data anomalies due to updates.

2.1 Automatic Updates of Data Warehouse - Level 0

- Take the Degree Dimension as an example:

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

- The Fact Table will have FKs by referencing PKs.
- The Fact Table will also have a composite PK which combines all the four dimension keys.

2.1 Automatic Updates of Data Warehouse - 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-hour day into three time periods.

2.1 Automatic Updates of Data Warehouse - Level 1

- We could make use of the Fact Table from Level-0 as the source for the TempFact Table on level 1.

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

2.1 Automatic Updates of Data Warehouse - Level 1

- 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 = …

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

update TempComputerLabFactLevel1 set TimeID = …
```

2.1 Automatic Updates of Data Warehouse - Level 1

- Create a database trigger to automatically trigger an insertion into the TempFact Table when there is an insertion to the Fact Level-0.

2.1 Automatic Updates of Data Warehouse - Level 1

```
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;
```

2.1 Automatic Updates of Data Warehouse - Level 1

```
  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;
```

2.1 Automatic Updates of Data Warehouse - Level 1

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

2.1 Automatic Updates of Data Warehouse - Level 1

```
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);
```

2.1 Automatic Updates of Data Warehouse - Level 1

```
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);
```

2.1 Automatic Updates of Data Warehouse - Level 1

- A database trigger must be created to ensure that for every record inserted into the TempFact Level1, it will increase the fact measure Number_of_Logins by one.

2.1 Automatic Updates of Data Warehouse - Level 1

```
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;
```

2.1 Automatic Updates of Data Warehouse - 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.

2.1 Automatic Updates of Data Warehouse - Level 2

```
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;
```

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.
- 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.

2.2 Expiry Date

- 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.
- This 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.

2.2 Expiry Date - 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)

2.2 Expiry Date - Level 0

```
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;
```

2.2 Expiry Date - Level 0

- 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.

2.2 Expiry Date - Level 0

```
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';
```

2.2 Expiry Date - Level 1

- The Level-1 star schema includes the creation of TempFact and Fact Tables.
- When trimming old data from the data warehouse, we must execute the ***delete*** command manually.

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

2.2 Expiry Date - Level 1

```
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;
```

2.2 Expiry Date - 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.

2.3 Data Warehouse Rules Changed

- 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.

2.3 Data Warehouse Rules Changed

- In the Lab Activities case study, Semester 1 is from 1-Jan to 15-Jul and Semester 2 is from 16-Jul to 31-Dec.
- Suppose the semester dates change. Instead of Semester 1 being from 1-Jan to 15-Jul, it is now from 1-Jan to 30-Jun, hence the two semesters are of equal length.

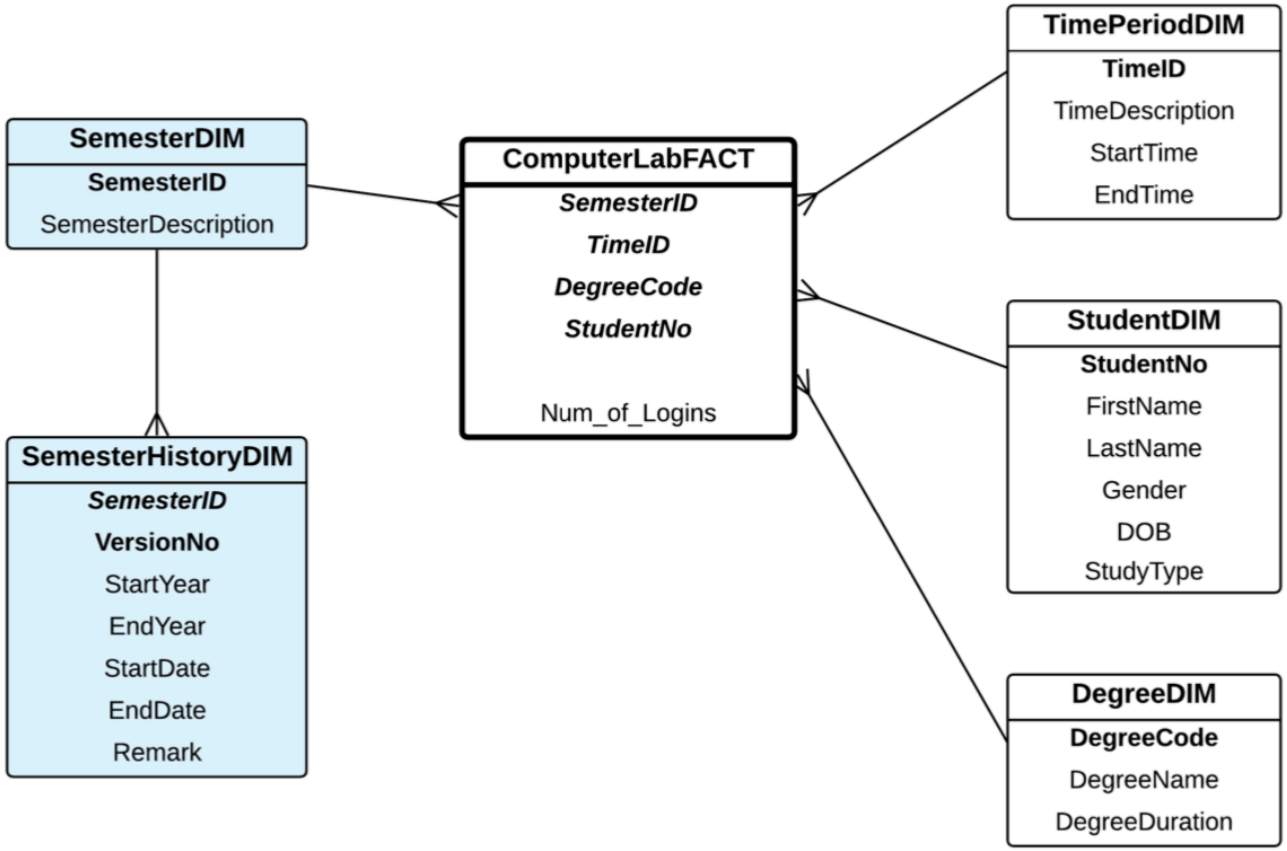
2.3 Data Warehouse Rules Changed - Level 0

- Since the Semester Dimension starts to exists from Level-1 upwards, 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.

2.3 Data Warehouse Rules Changed - Level 1

- Two places that will be affected by the change of the semester dates:
- 1) Semester Dimension: When the semester dates change, we can utilize a temporal dimension to store the history of the semester dates.

2.3 Data Warehouse Rules Changed - Level 1



2.3 Data Warehouse Rules Changed - Level 1

- 2) TempFact: 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.

2.3 Data Warehouse Rules Changed - Level 1

- 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.

2.3 Data Warehouse Rules Changed - 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.
- If Level-1 is updated correctly, the updates will be correctly reflected in Level-2 too.

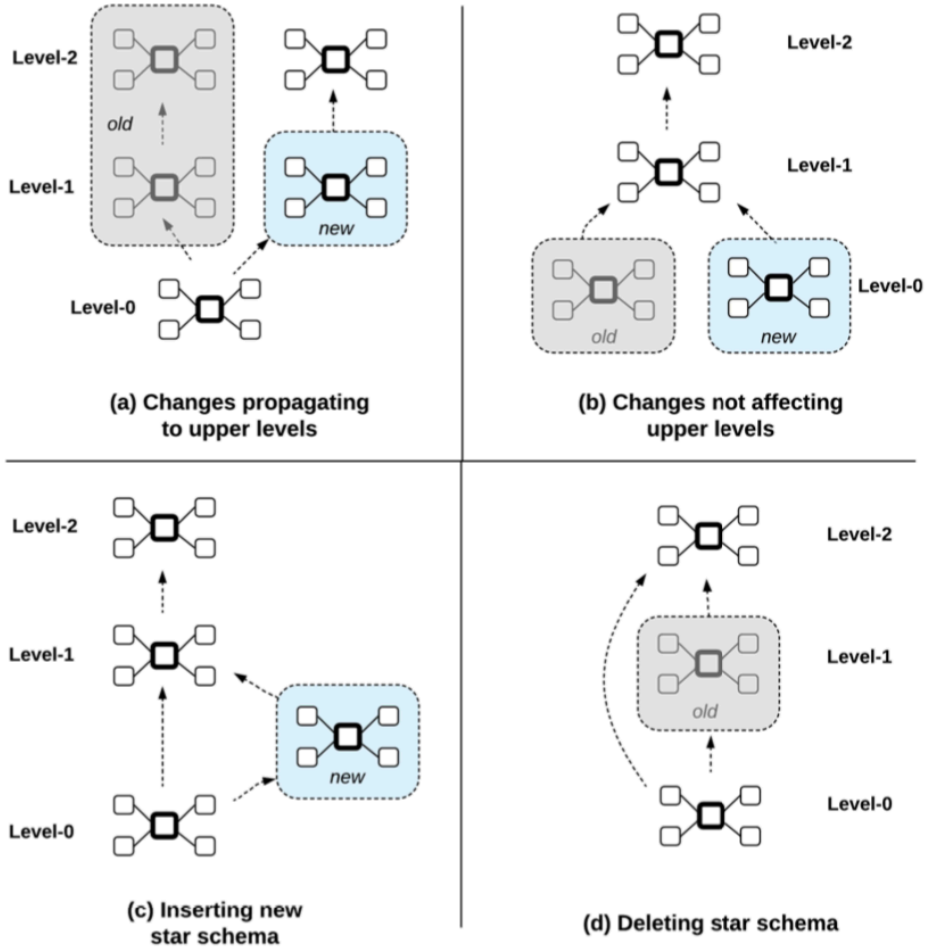
3 Data Warehousing Schema Evolution

- 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 known as **Data Warehousing Schema Evolution**.

3 Data Warehousing Schema Evolution

- There are four types of data warehousing schema evolution:
  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

3 Data Warehousing Schema Evolution



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.

3.1 Changes Propagating to the Next Levels

- 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.
- The 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.

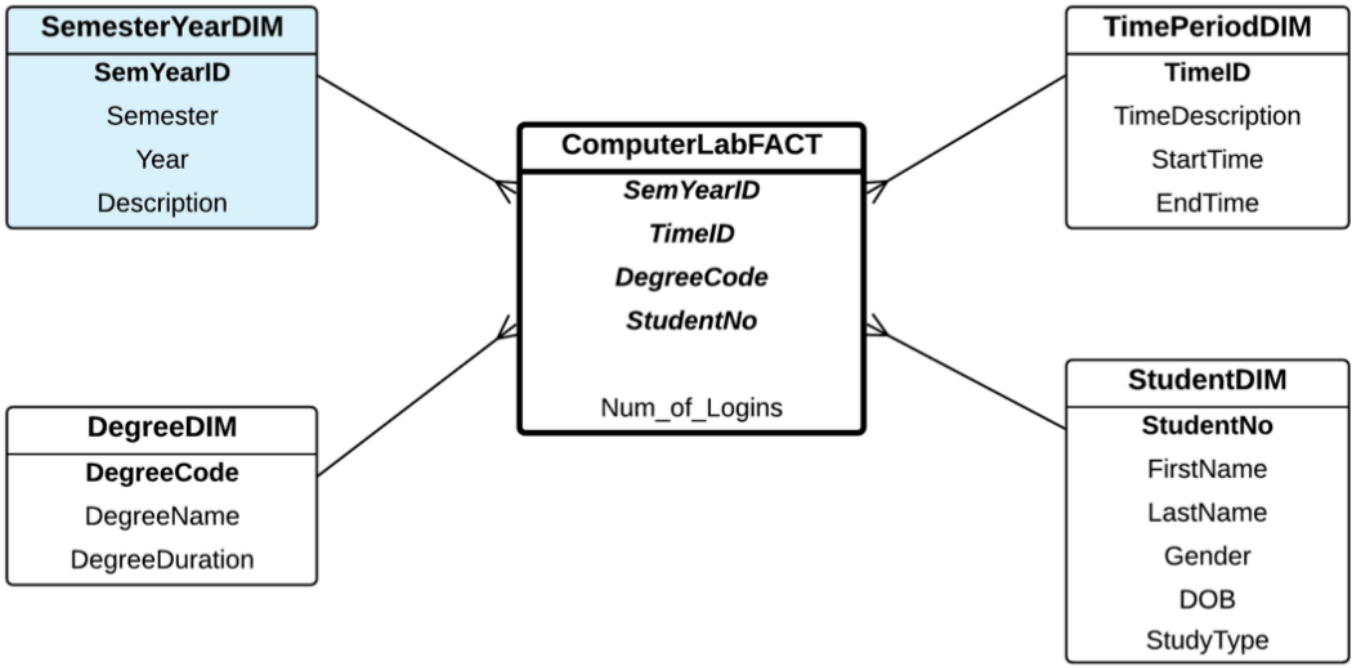
3.1 Changes Propagating to the Next Levels - Level 0

- The semester and year information will not appear until Level-1, so there are no changes to the Level-0 star schema.

3.1 Changes Propagating to the Next Levels - 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.

3.1 Changes Propagating to the Next Levels - Level 1



3.1 Changes Propagating to the Next Levels - Level 1

- 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.

3.1 Changes Propagating to the Next Levels - Level 1

```
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)
);
```

3.1 Changes Propagating to the Next Levels - Level 1

```
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');
```

3.1 Changes Propagating to the Next Levels - Level 1

```
update SemesterYearDimTemp
set SemYearID = Year || Semester;

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

3.1 Changes Propagating to the Next Levels - Level 1

- 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.

3.1 Changes Propagating to the Next Levels - Level 1

```
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;
```

3.1 Changes Propagating to the Next Levels - Level 1

```
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;
```

3.1 Changes Propagating to the Next Levels - Level 1

```
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;
```

3.1 Changes Propagating to the Next Levels - Level 1

- A TempFact Table can be created in the usual way but the Fact Table from Level-0 is re-used.

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

3.1 Changes Propagating to the Next Levels - Level 1

- 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));
```

3.1 Changes Propagating to the Next Levels - Level 1

```
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';
```

3.1 Changes Propagating to the Next Levels - Level 1

```
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';
```

3.1 Changes Propagating to the Next Levels - Level 1

- Converting the Login Time attribute to TimeID attribute.

```
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 = …
update TempComputerLabFactLevel1 set TimeID = …
```

3.1 Changes Propagating to the Next Levels - Level 1

- Finally, the database trigger to automatically trigger an insertion to the TempFact
- The final Fact Table for Level-1 is created by aggregating the four dimension identifiers and counting the StudentNo.

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

3.1 Changes Propagating to the Next Levels - Level 1

- 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.

3.1 Changes Propagating to the Next Levels - Level 1

```
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;
```

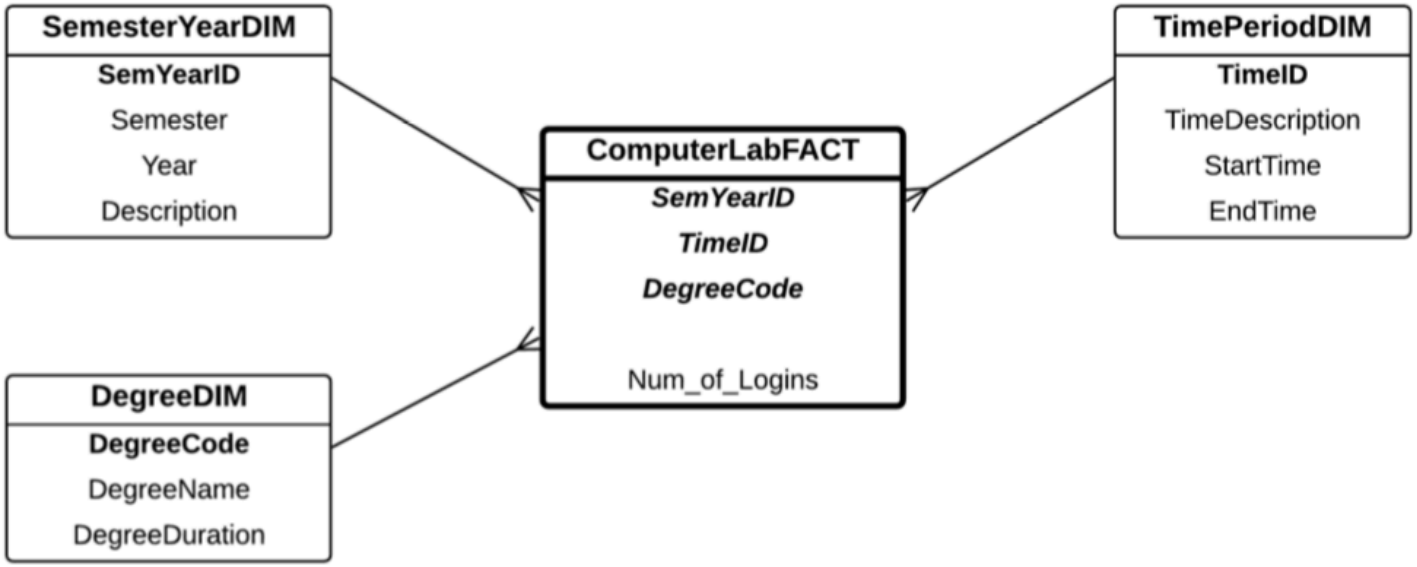
3.1 Changes Propagating to the Next Levels - Level 1

```
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;
```

3.1 Changes Propagating to the Next Levels - 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 Level-2 star schema basically removes the Student Dimension from Level-1.

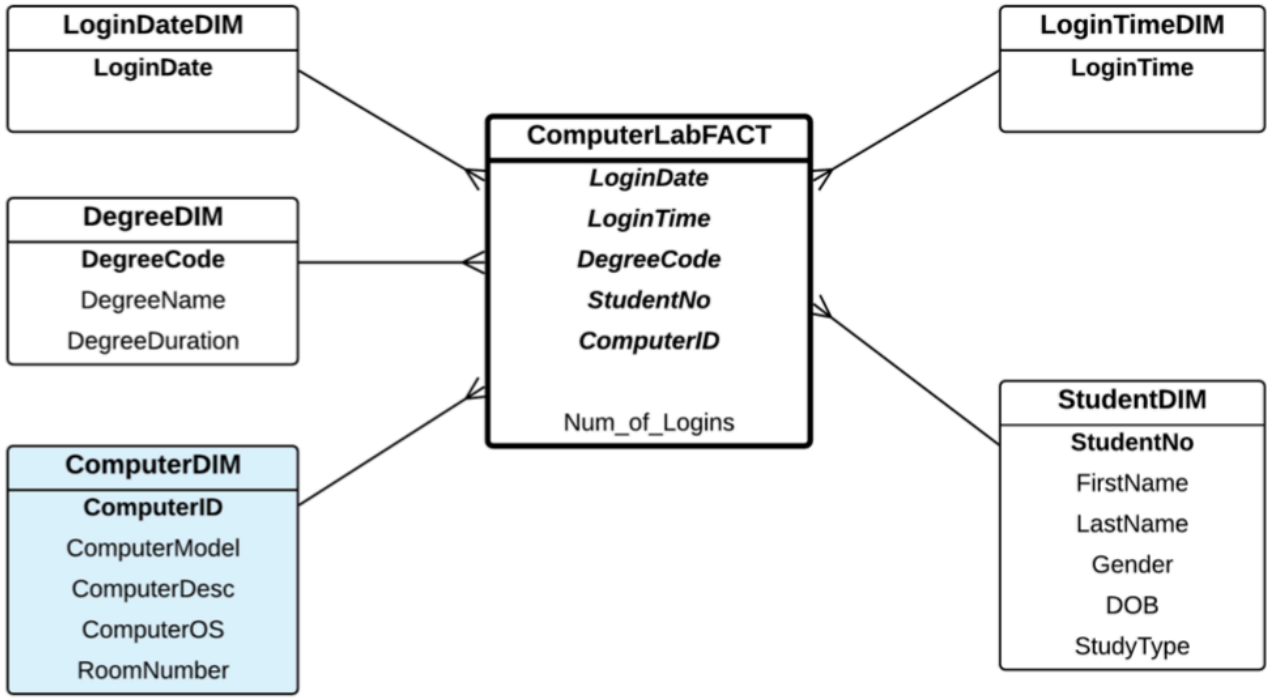
3.1 Changes Propagating to the Next Levels - Level 2



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.
- In the Lab Activities case study, 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.
- The other four dimensions are unchanged and the fact measure also remains the same.

3.2 Changes Not Affecting the Next Levels



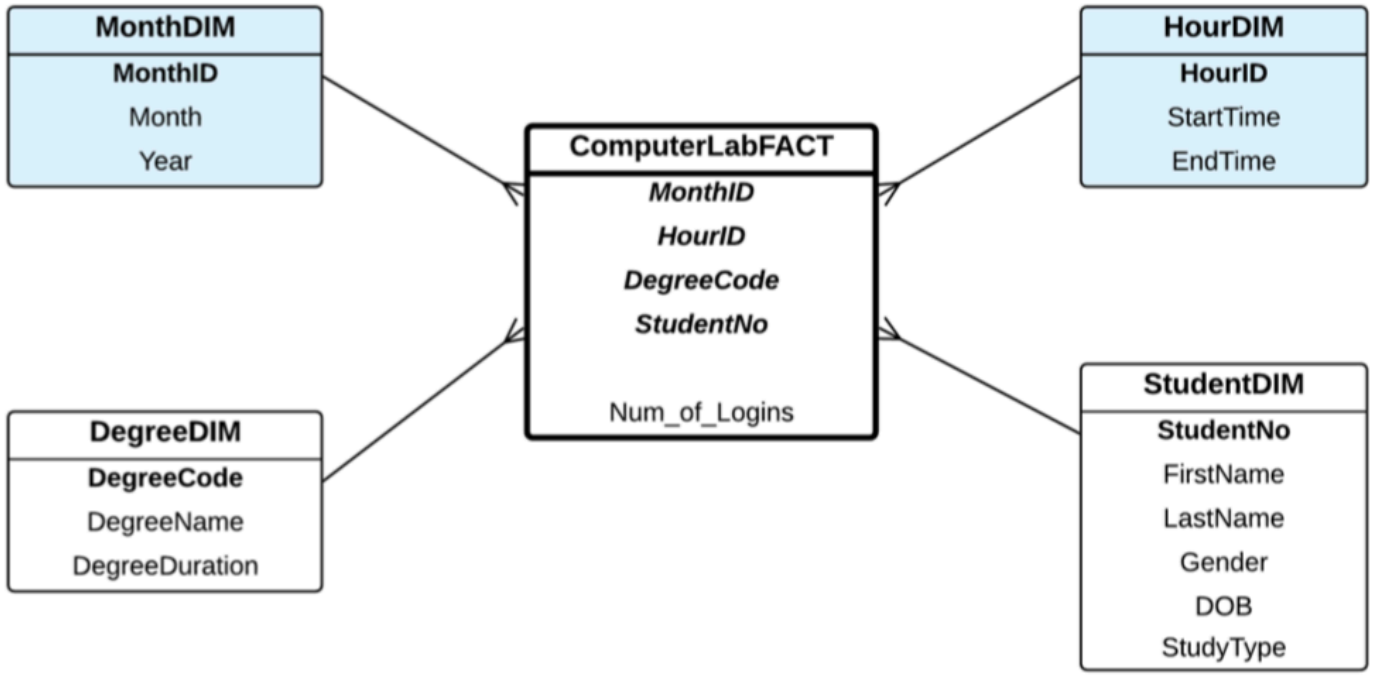
3.2 Changes Not Affecting the Next Levels

- 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, 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.

3.3 Inserting New Star Schema

- In the Lab Activities case study, 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 to insert a new star schema between these levels. Suppose the proposed star schema is at the month and hour granularity level.

3.3 Inserting New Star Schema



3.3 Inserting New Star Schema - Creation of the new star schema

- The creation of this new star schema could be done in the usual way, that is, the Month Dimension, and the Hour Dimension can be created manually.
- The other two dimensions, Degree and Student Dimensions, can be reused from the lower level.

3.3 Inserting New Star Schema - Creation of the new star schema

- For the Fact Table (extracts the records from the operational database)

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

3.3 Inserting New Star Schema - Creation of the new star schema

- For the Fact Table (extracts the records from the Level-0FactTable)

```
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;
```

3.3 Inserting New Star Schema - 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.
- The Hour Dimension does not need to be maintained manually because the dimension contains the 24-hour 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.

3.3 Inserting New Star Schema - 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.

- The TempFact was created based on the Fact Table from the Level-0 star schema. Therefore, there won't be any inter-dependency between the Level-1 star schema and the new Sub-Level-1 star schema.

3.3 Inserting New Star Schema - Impact on the next levels

- 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.

3.4 Deleting Star Schema

- In the Lab Activities case study, 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. 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.

3.4 Deleting 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. The fact measure is also the same, which is an aggregate value of Number_of_Logins.

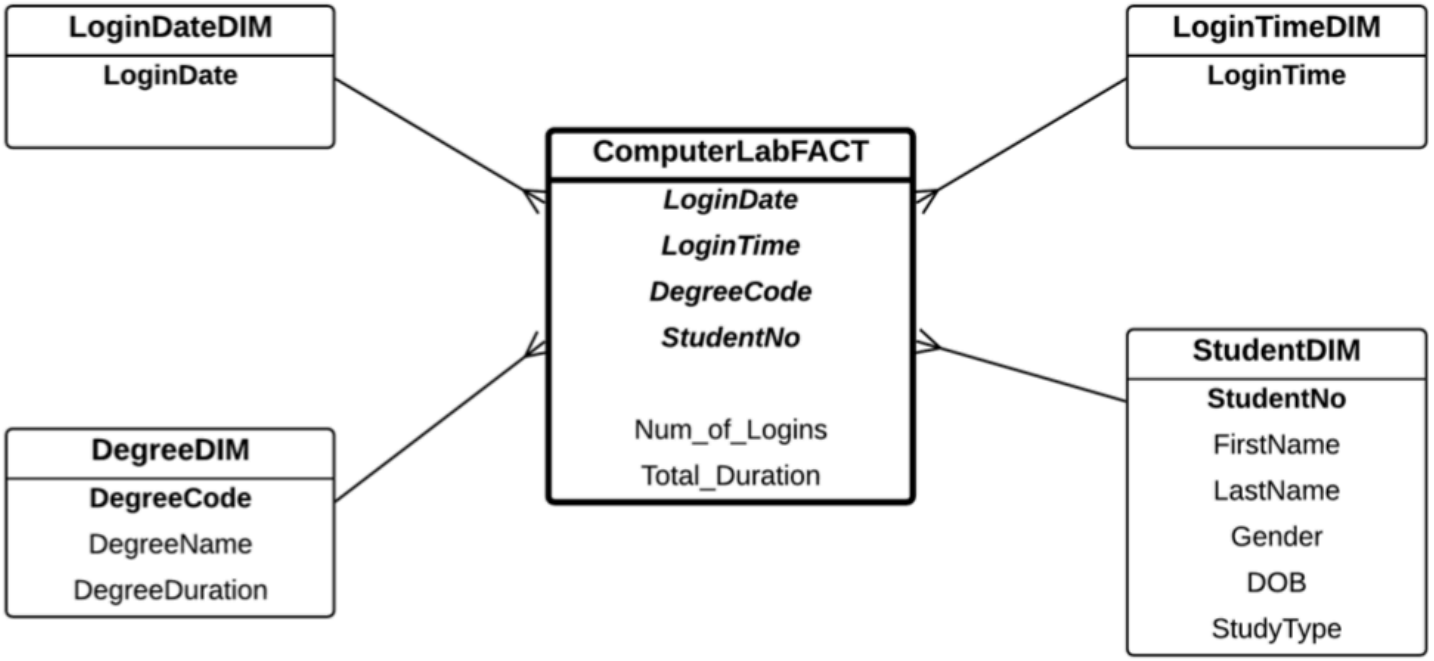
4 Operational Database Evolution

- As the operational database is changed, the data warehousing requirements may also change.

4.1 Changes in the Table Structure

- In 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.
- In this case, a new fact measure: Duration (or Total Duration), can be added to the star schema.

4.1 Changes in the Table Structure



4.1 Changes in the Table Structure

- There are two possible options to deal with this.

- Option 1 is to update the existing star schemas.

- Option 2 is to build a new data warehouse, since the star schema now has new requirements.

4.1 Changes in the Table Structure

- Option 1 : update the existing star schemas
- The existing Level-0 Fact Table needs to be altered by adding a new attribute

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

4.1 Changes in the Table Structure

- Option 1 : update the existing star schemas

- To handle the new records, the database trigger can be used

4.1 Changes in the Table Structure

```
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;
```

4.1 Changes in the Table Structure

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

4.1 Changes in the Table Structure

- Option 2 : build a new data warehouse

- The steps that need to be taken are the same as when building a new data warehouse

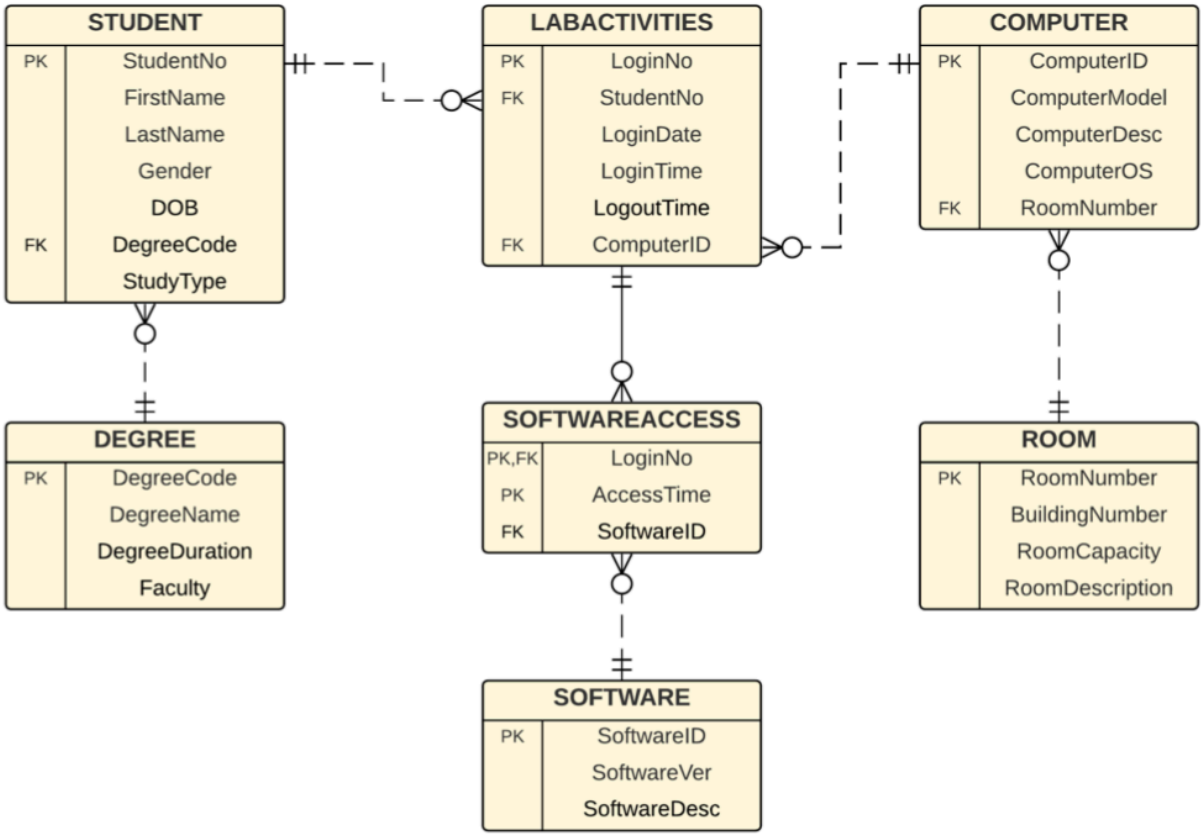
4.2 Changes in the E/R Schema

- Another type of change in the operational database is the addition of new entities (and hence new relationships) in the E/R diagram.
- Two areas of extension are applied in the Computer Lab Activities case study

- 1) Create an entity to store the Room information

- 2)Add Software Access and Software entities that related to the lab

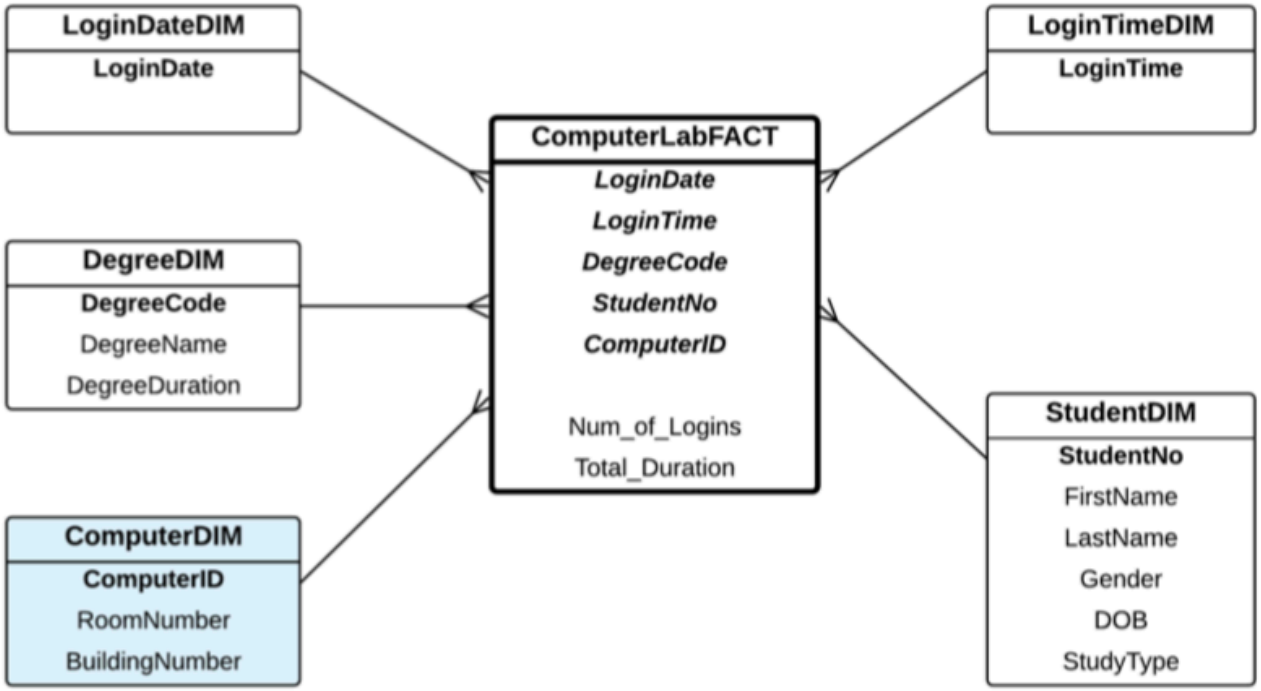
4.2 Changes in the E/R Schema



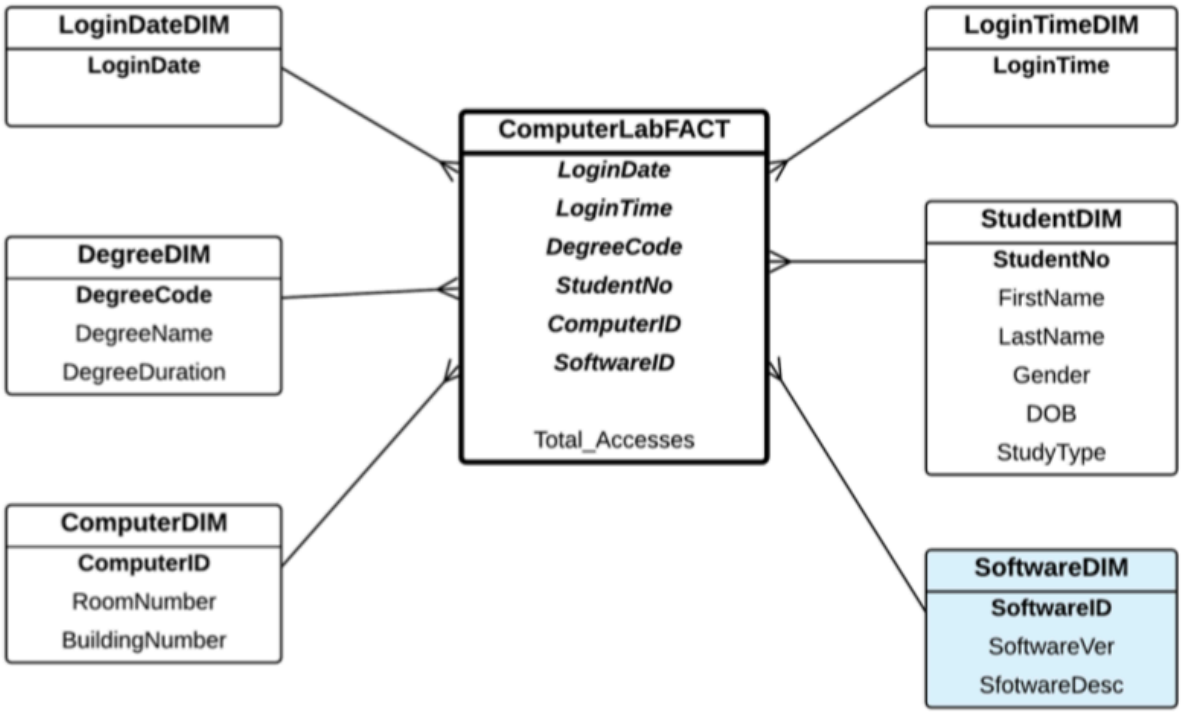
4.2 Changes in the E/R Schema

- The data warehouse is now expanded into two Level-0s.
- One is at the login granularity
- The second is at the software access granularity

4.2 Changes in the E/R Schema



4.2 Changes in the E/R Schema



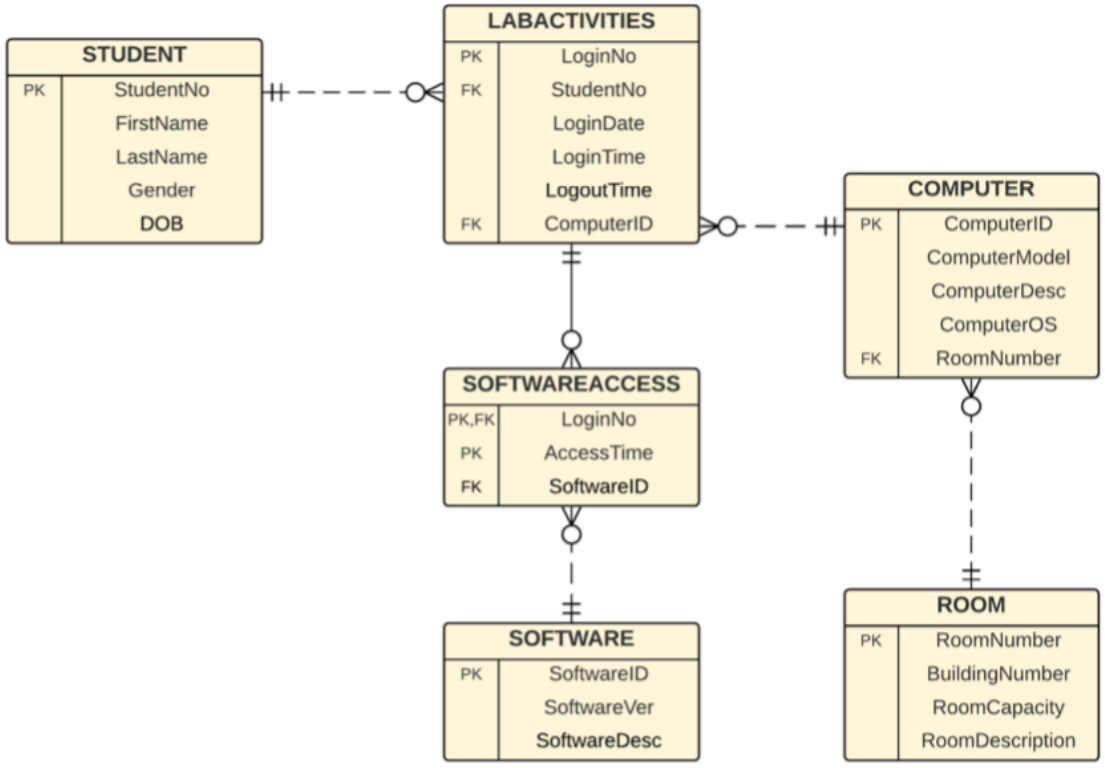
4.3 Changes in the Operational Database

- 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.

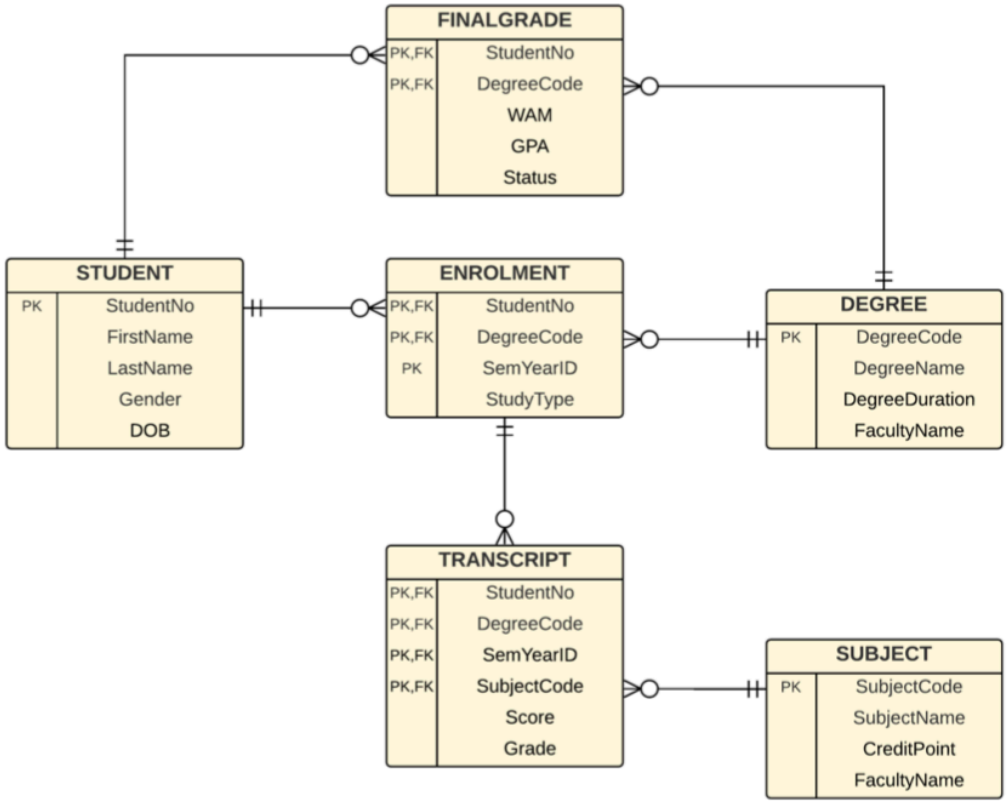
4.3 Changes in the Operational Database

- In the Lab Activities case study, suppose now the system that stores the computer lab activities is changed, then the Degree entity is no longer there.
- There is also a new system that keeps the enrolment records of students. It keeps track of every individual subject in which each student enrols.

4.3 Changes in the Operational Database

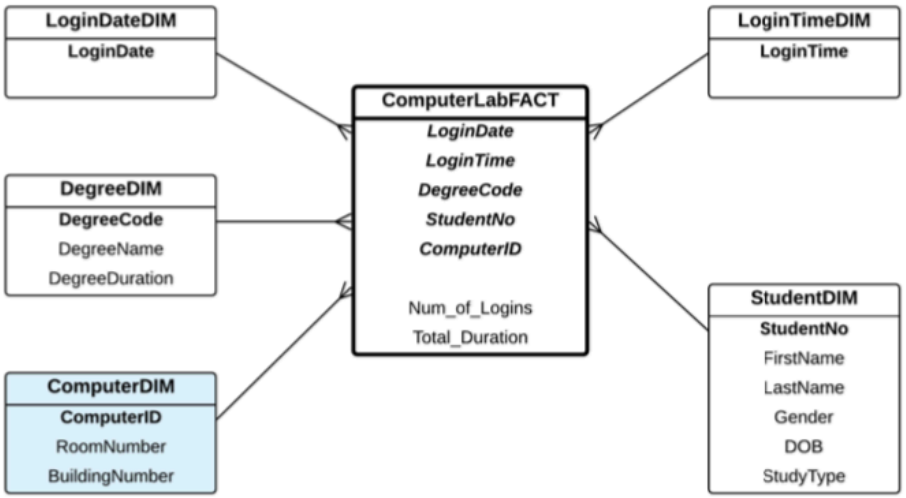


4.3 Changes in the Operational Database



4.3 Changes in the Operational Database

- The data warehouse has the star schema with five dimensions: Login Date, Login Time, Student, Degree, and Computer Dimensions, which is a Level-0 star schema.



4.3 Changes in the Operational Database

- There are two options to deal with Active Data Warehousing when the operational databases have changed.

- Option 1: Update existing star schema
- Option 2: Create new star schema

4.3 Changes in the Operational Database

- Option 1: Update existing star schema
- The Degree information needs to be taken from the new Student Enrolment database
- In the database trigger for the fact, it needs to obtain the degree information from the Student Enrolment system

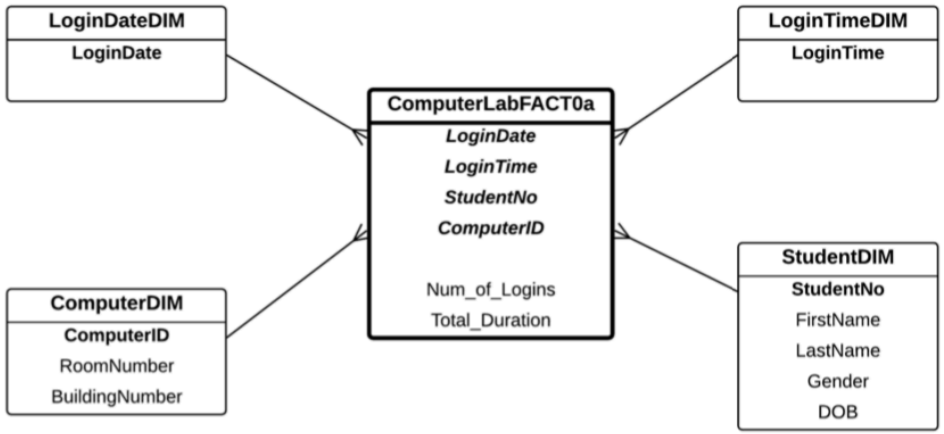
4.3 Changes in the Operational Database

- Option 2: Create new star schema
- Create a separate star schema from each operational database and after that, the two star schemas are combined

4.3 Changes in the Operational Database

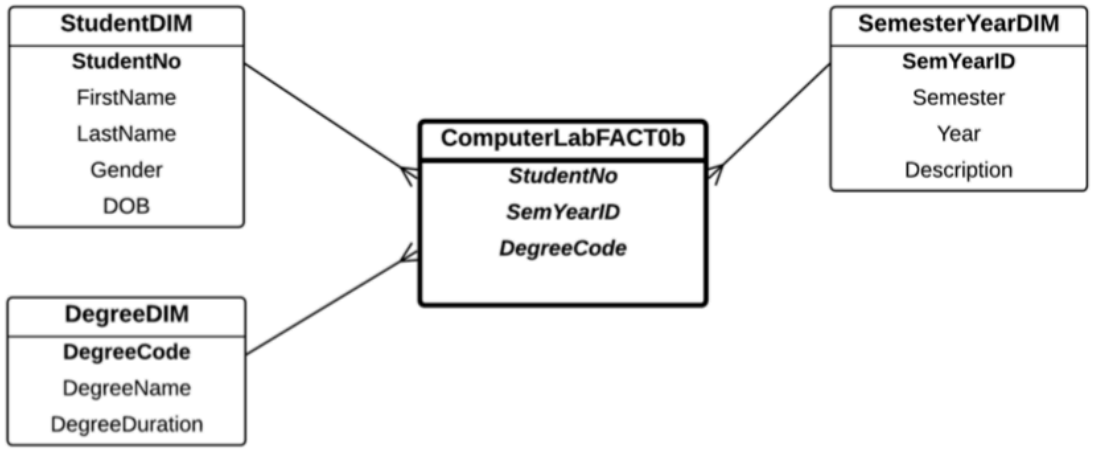
- Option 2: Create new star schema

- The first star schema is based on the Computer Lab Activities operational database



4.3 Changes in the Operational Database

- Option 2: Create new star schema
- The second star schema is based on the Student Enrolment operational database



4.3 Changes in the Operational Database

- Option 2: Create new star schema
- Once the two star schemas are created, they need to be merged to form the final star schema.
- The first step is to copy the first star schema to the final star schema, and add a new column for the Degree Code.

4.3 Changes in the Operational Database

- Option 2: Create new star schema

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

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

4.3 Changes in the Operational Database

- Option 2: Create new star schema
- 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.

4.3 Changes in the Operational Database

- Option 2: Create new star schema

```
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;
```

4.3 Changes in the Operational Database

- Option 2: Create new star schema

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

update ComputerLabFactNew0a
set SemYearID = Year || Semester;

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

4.3 Changes in the Operational Database

- Option 2: Create new star schema
- 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 schema.
- 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.

Summary

- Active Data Warehousing is solely about inter-dependency 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.
- There are three types of inter-dependencies studied in this chapter :

1) Incremental updates
2) Changes in the data warehouse
3) Changes in the operational database

- Since inter-dependency can be quite complex in many cases, Active Data Warehousing is used only for applications that really need active data in the data ware- house.