FIT5137 DATABASE ANALYSIS AND PROCESSING

Assignment

Group Member

27608174 ShuChen

27525260 Jie Pan

27356094 Dongyu Zhao

MONASH University
Information Technology

GROUP ASSIGNMENT COVER SHEET

Student ID Number	Surname	Given Names
27608174	Shu	Chen
27525260	Jie	Pan
27356094	Dongyu	Zhao

* Please include the names of all other group members.

Unit name and code	5137 Database analysis and processing	
Title of assignment	Assignment – Human Resources and Sales Order Warehouse	
Lecturer/tutor	David Taniar / Lingxiao Li	
Tutorial day and time	Thursday 6:00 pm	**Campus**
Is this an authorised group assignment?	**√ Yes** ☐ **No**	
Has any part of this assignment been previously submitted as part of another unit/course?	☐ **Yes** **√ No**	
Due Date 16/10/2017 23:55		**Date submitted 16/10/2017**

All work must be submitted by the due date. If an extension of work is granted this must be specified with the signature of the lecturer/tutor.

Extension granted until (date) ..... **Signature of lecturer/tutor .....**

Please note that it is your responsibility to retain copies of your assessments.

Intentional plagiarism or collusion amounts to cheating under Part 7 of the Monash University (Council) Regulations

Plagiarism: Plagiarism means taking and using another person's ideas or manner of expressing them and passing them off as one's own. For example, by failing to give appropriate acknowledgement. The material used can be from any source (staff, students or the internet, published and unpublished works).

Collusion: Collusion means unauthorised collaboration with another person on assessable written, oral or practical work and includes paying another person to complete all or part of the work.

Where there are reasonable grounds for believing that intentional plagiarism or collusion has occurred, this will be reported to the Associate Dean (Education) or delegate, who may disallow the work concerned by prohibiting assessment or refer the matter to the Faculty Discipline Panel for a hearing.

Student Statement:

- I have read the university's Student Academic Integrity Policy and Procedures.
- I understand the consequences of engaging in plagiarism and collusion as described in Part 7 of the Monash University (Council) Regulations http://adm.monash.edu/legal/legislation/statutes
- I have taken proper care to safeguard this work and made all reasonable efforts to ensure it could not be copied.
- No part of this assignment has been previously submitted as part of another unit/course.
- I acknowledge and agree that the assessor of this assignment may for the purposes of assessment, reproduce the assignment and:
  - i. provide to another member of faculty and any external marker; and/or
  - ii. submit it to a text matching software; and/or
  - iii. submit it to a text matching software which may then retain a copy of the assignment on its database for the purpose of future plagiarism checking.
- I certify that I have not plagiarised the work of others or participated in unauthorised collaboration when preparing this assignment.

Signature ...Shu Chen, Jie Pan, Dongyu Zhao..... Date.....10/16/2017.....

* delete (iii) if not applicable

Signature ___Shu Chen_____Date:_____10/16/2017___

Signature ___Jie Pan_____Date:_____ 10/16/2017___

Signature ___Dongyu Zhao_____Date:_____ 10/16/2017___

Privacy Statement
The information on this form is collected for the primary purpose of assessing your assignment and ensuring the academic integrity requirements of the University are met. Other purposes of collection include recording your plagiarism and collusion declaration, attending to course and administrative matters and statistical analyses. If you choose not to complete all the questions on this form it may not be possible for Monash University to assess your assignment. You have a right to access personal information that Monash University holds about you, subject to any exceptions in relevant legislation. If you wish to seek access to your personal information or inquire about the handling of your personal information, please contact the University Privacy Officer: privacyofficer@adm.monash.edu.au

Updated: 17 Jun 2014

Details of ORACLE accounts:

ID: S27608174

Password: -3-2b2b2b

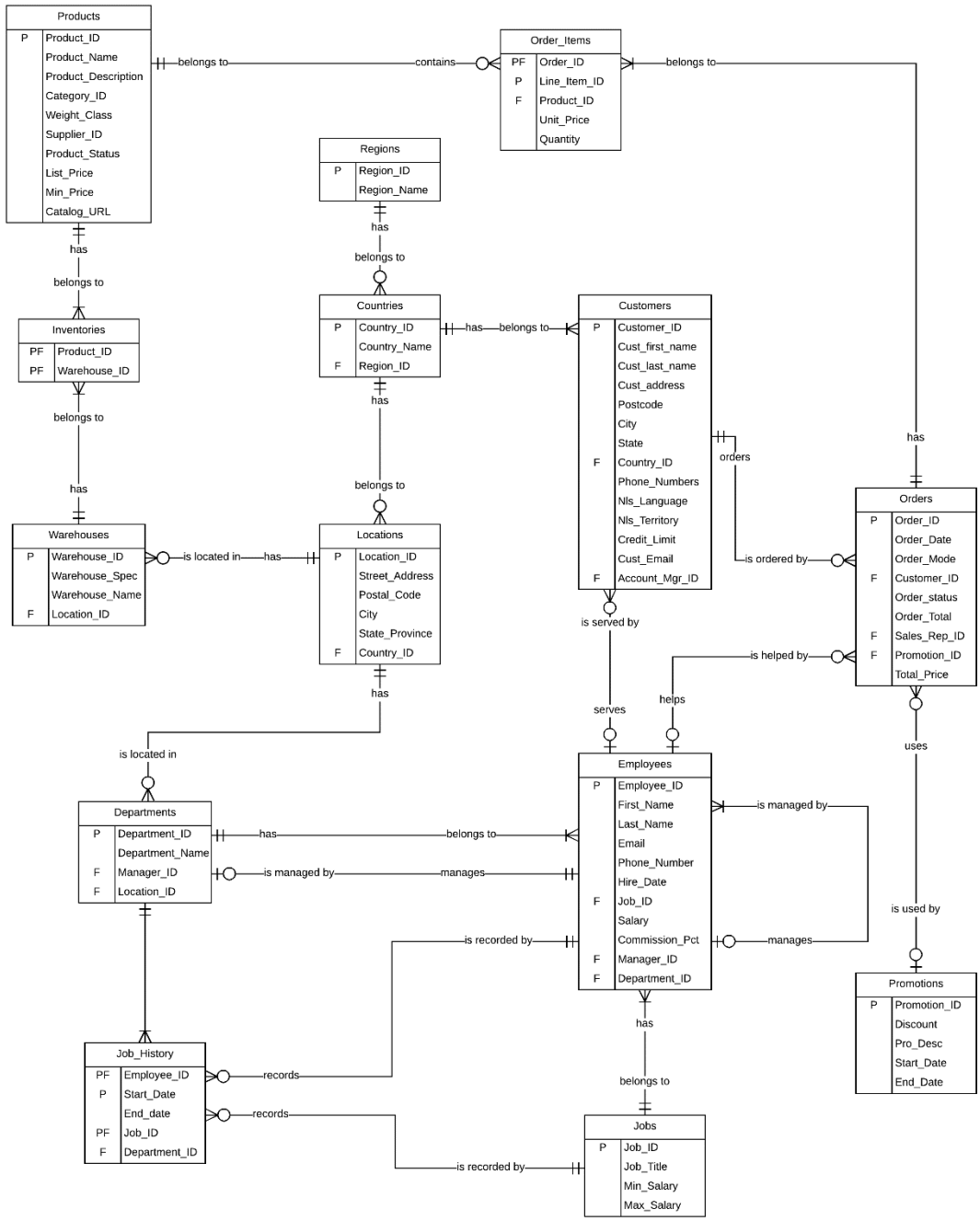
Percentage of Contribution

Name	ID	Contribution	Parts
Shu Chen	27608174	33%	Task 1 & Task 5
Jie Pan	27525260	33%	Task 2.c, Task 3.c, d, Task 4.b, d
Dongyu Zhao	27356094	33%	Task 2.a, b, Task 3.a, b, Task 4.a, c

Contents

Task 1 E/R diagram & Data Cleaning

(a) E/R diagram



(b) Data cleaning

1. Manager who is not in employees

Detect error

```
SELECT DISTINCT
    manager_id
FROM
    employees
WHERE
    manager_id NOT IN (
        SELECT
            employee_id
        FROM
            employees
    );
```

Data cleaning

```
UPDATE employees
    SET
        manager_id = 101
WHERE
    manager_id = 210;
```

Before data cleaning

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
98	197	Kevin	Feeney	KFEENEY	650.507.9822	23-MAY-06	SH_CLERK	3000	(null)	124	50
99	198	Donald	OConnell	DOCONNEL	650.507.9833	21-JUN-07	SH_CLERK	2600	(null)	124	50
100	199	Douglas	Grant	DGRANT	650.507.9844	13-JAN-08	SH_CLERK	2600	(null)	124	50
101	200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-03	AD_ASST	4400	(null)	101	10
102	201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-04	MK_MAN	13000	(null)	100	20
103	202	Pat	Fay	PFAY	603.123.6666	17-AUG-05	MK_REP	6000	(null)	210	20
104	203	Susan	Mavris	SMAVRIS	515.123.7777	07-JUN-02	HR_REP	6500	(null)	101	40
105	204	Hermann	Baer	HBAER	515.123.8888	07-JUN-02	PR_REP	10000	(null)	101	70

After data cleaning

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
98	197	Kevin	Feeney	KFEENEY	650.507.9822	23-MAY-06	SH_CLERK	3000	(null)	124	50
99	198	Donald	OConnell	DOCONNEL	650.507.9833	21-JUN-07	SH_CLERK	2600	(null)	124	50
100	199	Douglas	Grant	DGRANT	650.507.9844	13-JAN-08	SH_CLERK	2600	(null)	124	50
101	200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-03	AD_ASST	4400	(null)	101	10
102	201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-04	MK_MAN	13000	(null)	100	20
103	202	Pat	Fay	PFAY	603.123.6666	17-AUG-05	MK_REP	6000	(null)	101	20
104	203	Susan	Mavris	SMAVRIS	515.123.7777	07-JUN-02	HR_REP	6500	(null)	101	40
105	204	Hermann	Baer	HBAER	515.123.8888	07-JUN-02	PR_REP	10000	(null)	101	70

2. Salary is negative

Detect error

```
SELECT
    *
FROM
    employees
WHERE
    salary < 0;
```

Data cleaning

```
UPDATE employees
    SET
        salary = 4800
WHERE
    employee_id = 106;
```

Before data cleaning

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	106	Valli	Pataballa	VPATABAL	590.423.4560	05-FEB-06	IT_PROG	-4800	(null)	103	60

After data cleaning

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	106	Valli	Pataballa	VPATABAL	590.423.4560	05-FEB-06	IT_PROG	4800	(null)	103	60

3. Employee's manager is null

Detect error

```
SELECT
    *
FROM
    employees
WHERE
    manager_id IS NULL;
```

Data cleaning

```
UPDATE employees
    SET
        manager_id = 100
WHERE
    employee_id = 100;
```

Before data cleaning

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	100	Steven	King	SKING	515.123.4567	17-JUN-03	AD_PRES	24000	(null)	(null)	90

After data cleaning

	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	100	Steven	King	SKING	515.123.4567	17-JUN-03	AD_PRES	24000	(null)	100	90

4. Duplicate Country_id in Countries

Detect error

```
SELECT
    country_id,
    COUNT(*)
FROM
    countries
GROUP BY
    country_id
HAVING
    COUNT(*) > 1;

SELECT
    *
FROM
    countries
WHERE
        country_id = 'DK'
    OR
        country_id = 'ML';
```

Data cleaning

```
DROP TABLE countries;

CREATE TABLE countries
    AS
        SELECT DISTINCT
            *
        FROM
            hosales.countries;
```

Before data cleaning

	COUNTRY_ID	COUNTRY_NAME	REGION_ID
1	DK	Denmark	1
2	DK	Denmark	1
3	ML	Malaysia	3
4	ML	Malaysia	3

After data cleaning

	COUNTRY_ID	COUNTRY_NAME	REGION_ID
1	ML	Malaysia	3
2	DK	Denmark	1

5. Duplicate Customer_id in Customers

Detect error

```
SELECT
    customer_id,
    COUNT(*)
FROM
    customers
GROUP BY
    customer_id
HAVING
    COUNT(*) > 1;

SELECT
    *
FROM
    customers
WHERE
    customer_id = 101;
```

Data cleaning

```
DROP TABLE customers;

CREATE TABLE customers
    AS
        SELECT DISTINCT
            *
        FROM
            hosales.customers;
```

Before data cleaning

	CUSTOMER_ID	CUST_FIRST_NAME	CUST_LAST_NAME	CUST_ADDRESS	POSTCODE	CITY	STATE	COUNTRY_ID	PHONE_NUMBERS	NLS_LANGUAGE	NLS_TERRIT
1	101	Constantin	Welles	514 W Superior St	46901	Kokomo	IN	US	+1 317 123 4104	us	AMERICA
2	101	Constantin	Welles	514 W Superior St	46901	Kokomo	IN	US	+1 317 123 4104	us	AMERICA

After data cleaning

	CUSTOMER_ID	CUST_FIRST_NAME	CUST_LAST_NAME	CUST_ADDRESS	POSTCODE	CITY	STATE	COUNTRY_ID	PHONE_NUMBERS	NLS_LANGUAGE	NLS_TERRIT
1	101	Constantin	Welles	514 W Superior St	46901	Kokomo	IN	US	+1 317 123 4104	us	AMERICA

6. Products table list_price smaller than min_price

Detect error

```
SELECT
    *
FROM
    products
WHERE
    list_price < min_price;
```

Data cleaning

```
UPDATE products
    SET
        list_price = 337
WHERE
```

```
    product_id = 3400;

UPDATE products
    SET
        min_price = 328
WHERE
    product_id = 3400;
```

Before data cleaning

ID	PRODUCT_NAME	PRODUCT_DESC...	CATEGORY_ID	WEIGHT_CLASS	SUPPLIER_ID	PRODUCT_STATUS	LIST_PRICE	MIN_PRICE	CATALOG_URL
1 00	HD 8GB /SE	8GB capacity...	13	2	102063	orderable	328	337	http://www.supp-102

After data cleaning

	PRODUCT_ID	PRODUCT_NAME	PRODUCT_DESC...	CATEGORY_ID	WEIGHT_CLASS	SUPPLIER_ID	PRODUCT_STATUS	LIST_PRICE	MIN_PRICE	CATALOG_URL
1	3400	HD 8GB /SE	8GB capacit...	13	2	102063	orderable	337	328	http://www.supp-1

7. Duplicate Order_id in Order table

Detect error

```
SELECT
    order_id,
    COUNT(*)
FROM
    orders
GROUP BY
    order_id
HAVING
    COUNT(*) > 1;

SELECT
    *
FROM
    orders
WHERE
    order_id = 2359;
```

Data cleaning

```
DROP TABLE orders;

CREATE TABLE orders
    AS
        SELECT DISTINCT
            *
        FROM
            hosales.orders;

UPDATE orders
    SET
        order_id = 2463
WHERE
        order_id = 2359
    AND
        order_mode = 'online';
```

Before data cleaning

	ORDER_ID	ORDER_DATE	ORDER_MODE	CUSTOMER_ID	ORDER_STATUS	ORDER_TOTAL	SALES_REP_ID	PROMOTION_ID	TOTAL_PRICE
1	2359	08-JAN-06 10.34.13.112233000 PM	online	106	9	5542.1	(null)	1	5542.1
2	2359	01-JAN-08 06.03.12.654278000 PM	direct	104	2	762	109	1	762

After data cleaning

	ORDER_ID	ORDER_DATE	ORDER_MODE	CUSTOMER_ID	ORDER_STATUS	ORDER_TOTAL	SALES_REP_ID	PROMOTION_ID	TOTAL_PRICE
1	2359	01-JAN-08 06.03.12.654278000 PM	direct	104	2	762	109	1	762

	ORDER_ID	ORDER_DATE	ORDER_MODE	CUSTOMER_ID	ORDER_STATUS	ORDER_TOTAL	SALES_REP_ID	PROMOTION_ID	TOTAL_PRICE
1	2463	08-JAN-06 10.34.13.112233000 PM	online	106	9	5542.1	(null)	1	5542.1

8. Duplicate Product_id and Warehouse_id in Inventories table

Detect error

```
SELECT
    product_id,
    warehouse_id,
    COUNT(*)
FROM
    inventories
GROUP BY
    product_id,
    warehouse_id
HAVING
    COUNT(*) > 1;


SELECT
    *
FROM
    inventories
WHERE
        product_id = 2402
    AND
        warehouse_id = 2
    OR
        product_id = 2419
    AND
        warehouse_id = 2
    OR
        product_id = 2322
    AND
        warehouse_id = 1
    OR
        product_id = 3300
    AND
        warehouse_id = 2
    OR
        product_id = 3086
    AND
        warehouse_id = 4;
```

Data cleaning

```
DROP TABLE inventories;

CREATE TABLE inventories
    AS
        SELECT DISTINCT
            *
        FROM
            hosales.inventories;
```

Before data cleaning

	PRODUCT_ID	WAREHOUSE_ID
1	2322	1
2	2322	1
3	2402	2
4	2419	2
5	3300	2
6	2402	2
7	2419	2
8	3300	2
9	3086	4
10	3086	4

After data cleaning

	PRODUCT_ID	WAREHOUSE_ID
1	2402	2
2	2419	2
3	2322	1
4	3300	2
5	3086	4

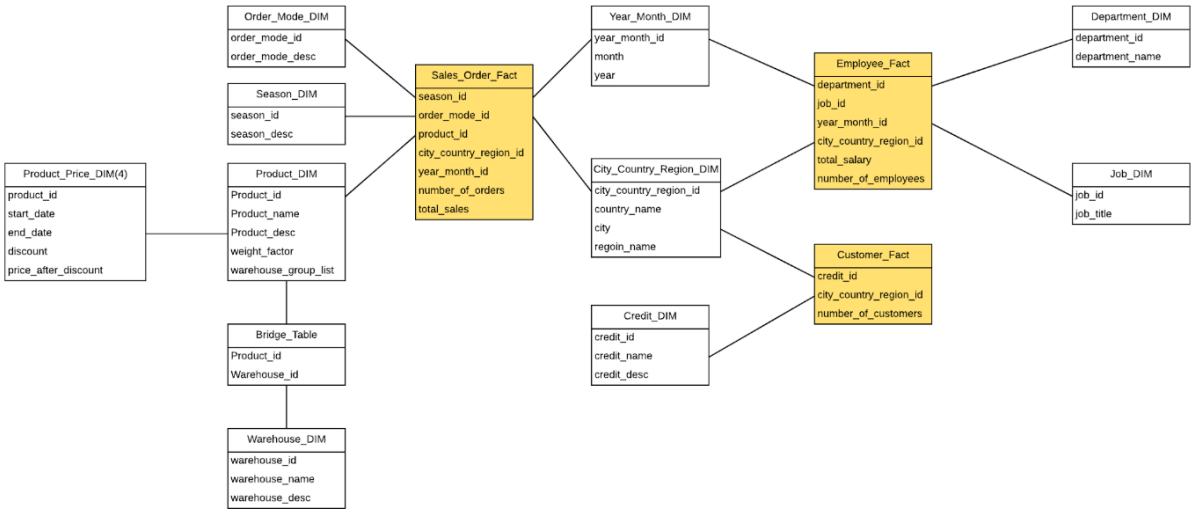
Task2 Multi-fact star schemas

(a) Two additional queries.

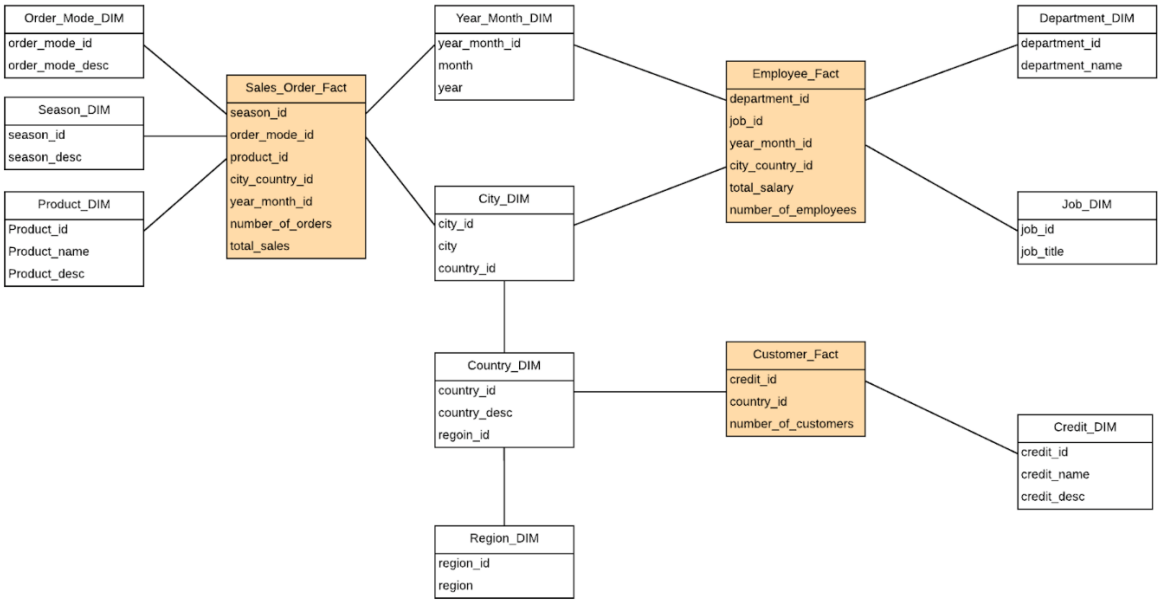
1.What is the total sales by each mode and each year?
2.What is the number of employees by each department and by each country?

(b) Two versions of star schema diagrams and details table

Version-1:



Version-2:



	Version-1	Version-2
Hierarchy		Employee_Fact, Sales_Order_Fact, Customer_Fact
Bridge Table	Sales_Order_Fact	
Temporal	Sales_Order_Fact (SCD Type 4)	Sales_Order_Fact (SCD Type 0)

(c) A short explanation for each of the star schemas.

In star schema version-1, because the Sales_Order_Fact is not directly link with the Warehouse_DIM, the Bridge_Table is used to link Product_DIM and Warehouse_DIM. Weight factor in Product_DIM is a proportion of the product that is from each warehouse for that product.

An attribute called “warehouse_group_list” is used keep all warehouses for each product. Product_Price_DIM is used to store the price history according to different time periods.

There is no hierarchy in star schema version-1. In star schema version-2, all the fact tables directly link with dimension tables. The hierarchy is country, region, city dimensions.

Task3 Implement star schemas

Implement environment (after data cleaning)

```
create table employees as select * from hosales.employees;
create table departments as select * from hosales.departments;
create table jobs as select * from hosales.jobs;
create table job_history as select * from hosales.job_history;
create table regions as select * from hosales.regions;
create table countries as select distinct * from hosales.countries;
create table locations as select * from hosales.locations;
create table customers as select distinct * from hosales.customers;
create table products as select * from hosales.products;
create table orders as select distinct * from hosales.orders;
create table order_items as select * from hosales.order_items;
create table warehouses as select * from hosales.warehouses;
create table inventories as select distinct * from hosales.inventories;
create table promotions as select * from hosales.promotions;

update employees
set manager_id = 101
where manager_id = 210;

update employees
set salary = 4800
where employee_id = 106;

update employees
set department_id = 80
where employee_id = 178;

update employees
set manager_id = 100
where employee_id = 100;

update products
set list_price = 337
where product_id = 3400;

update products
set min_price = 328
where product_id = 3400;
```

```
update orders
set order_id = 2463
where order_id = 2359 and order_mode = 'online';
```

(a) SQL statements to create the star schemas Version-1.

```
-- order_mode_dim
create table order_mode_dim (order_mode_id char(1), order_mode_desc varchar(20));
insert into order_mode_dim values ('O', 'online');
insert into order_mode_dim values ('D', 'direct');

-- season_dim
create table season_dim (season_id int, season_desc varchar(10));
insert into season_dim values (1, 'Spring');
insert into season_dim values (2, 'Summer');
insert into season_dim values (3, 'Autumn');
insert into season_dim values (4, 'Winter');

-- product_dim
create table product_dim as
    select p.product_id, p.product_name, p.product_description as product_desc,
        1/count(*) as weight_factor,
        nvl(listagg(i.warehouse_id, '_') within group (order by i.warehouse_id), 'UNKNOWN') as
warehouse_group_list
    from products p, inventories i
    where p.product_id = i.product_id (+)
    group by p.product_id, p.product_name, p.product_description;

-- bridge_table
create table bridge_table as select * from inventories;

-- warehouse_dim
create table warehouse_dim as select * from warehouses;

-- tmp_promotions
create table tmp_promotions as select discount, start_date, end_date from promotions;

delete from tmp_promotions where start_date is null;
insert into tmp_promotions values
('Full Price',to_date('20040101','yyyymmdd'),to_date('20070531','yyyymmdd'));
insert into tmp_promotions values
('Full Price',to_date('20070701','yyyymmdd'),to_date('20071109','yyyymmdd'));
insert into tmp_promotions values
```

```
('Full Price',to_date('20071112','yyyymmdd'),to_date('20071130','yyyymmdd'));
insert into tmp_promotions values
('Full Price',to_date('20080101','yyyymmdd'),to_date('20080531','yyyymmdd'));
insert into tmp_promotions values
('Full Price',to_date('20080701','yyyymmdd'),to_date('20081231','yyyymmdd'));

alter table tmp_promotions add discount_value numeric(3,2);

update tmp_promotions set discount_value = 0 where upper(discount) = 'FULL PRICE';
update tmp_promotions set discount_value = 0.3 where upper(discount) = '30% OFF';
update tmp_promotions set discount_value = 0.2 where upper(discount) = '20% OFF';
update tmp_promotions set discount_value = 0.1 where upper(discount) = '10% OFF';

-- product_price_dim
create table product_price_dim as
   select    oi.product_id, p.start_date, p.end_date, p.discount, oi.unit_price * (1 - p.discount_value) as
price_after_discount
   from (select distinct product_id, unit_price from order_items) oi, tmp_promotions p;

-- year
create table year(year char(4));

insert into year values('2004');
insert into year values('2005');
insert into year values('2006');
insert into year values('2007');
insert into year values('2008');

-- month
create table month(month char(2));

insert into month values('01');
insert into month values('02');
insert into month values('03');
insert into month values('04');
insert into month values('05');
insert into month values('06');
insert into month values('07');
insert into month values('08');
insert into month values('09');
insert into month values('10');
insert into month values('11');
insert into month values('12');
```

```
-- all_months (join year and month)
create table all_months as select y.year || m.month as year_month_id, y.year, m.month from month m,
year y;

-- month_year_dim
create table year_month_dim as select year || month as year_month_id, year, month from
   (select distinct to_char(order_date, 'yyyy') as year, to_char(order_date, 'mm') as month from orders
   union
   select distinct year, month from all_months);

-- city_country_regoin_dim
create table city_country_region_dim as select city || '-' || country_id || '-' || region_id as
city_country_region_id, city, country_name, region_name
from
(select distinct cu.city, co.country_id, r.region_id, co.country_name, r.region_name
from customers cu, countries co, regions r
where cu.country_id = co.country_id and co.region_id = r.region_id
union
select distinct l.city, co.country_id, r.region_id, co.country_name, r.region_name
from locations l, countries co, regions r
where l.country_id = co.country_id and co.region_id = r.region_id);

-- credit_dim
create table credit_dim (credit_id char(1), credit_name varchar(50), credit_desc varchar(255));
insert into credit_dim values ('L', 'low credit', 'credit <= 1500');
insert into credit_dim values ('M', 'medium credit', '1500 < credit <= 3500');
insert into credit_dim values ('H', 'high credit', 'credit > 3500');

-- department_dim
create table department_dim as select department_id, department_name from departments;

-- job_dim
create table job_dim as select job_id, job_title from jobs;

-- tmp_sales_order_fact
create table tmp_sales_order_fact as
   select o.order_date, o.order_mode, oi.product_id, c.city, co.country_id, co.region_id, oi.unit_price,
oi.quantity, p.discount
   from orders o, order_items oi, customers c, promotions p, countries co
   where o.order_id = oi.order_id and o.customer_id = c.customer_id and o.promotion_id = p.promotion_id
and co.country_id = c.country_id;

alter table tmp_sales_order_fact add discount_value numeric(3,2);
update tmp_sales_order_fact set discount_value = 0 where upper(discount) = 'FULL PRICE';
```

```
update tmp_sales_order_fact set discount_value = 0.3 where upper(discount) = '30% OFF';
update tmp_sales_order_fact set discount_value = 0.2 where upper(discount) = '20% OFF';
update tmp_sales_order_fact set discount_value = 0.1 where upper(discount) = '10% OFF';

alter table tmp_sales_order_fact add season_id int;
update tmp_sales_order_fact set season_id = 1 where to_char(order_date, 'mm') >= '09' and
to_char(order_date, 'mm') <= '11';
update tmp_sales_order_fact set season_id = 2 where to_char(order_date, 'mm') >= '12' or
to_char(order_date, 'mm') <= '02';
update tmp_sales_order_fact set season_id = 3 where to_char(order_date, 'mm') >= '03' and
to_char(order_date, 'mm') <= '05';
update tmp_sales_order_fact set season_id = 4 where to_char(order_date, 'mm') >= '06' and
to_char(order_date, 'mm') <= '08';

alter table tmp_sales_order_fact add order_mode_id char(1);
update tmp_sales_order_fact set order_mode_id = 'O' where upper(order_mode) = 'ONLINE';
update tmp_sales_order_fact set order_mode_id = 'D' where upper(order_mode) = 'DIRECT';

-- sales_order_fact
create table sales_order_fact as
   select season_id, order_mode_id, product_id,
      city || '-' || country_id || '-' || region_id as city_country_region_id,
   to_char(order_date, 'yyyymm') as year_month_id, count(*) as number_of_orders, sum(quantity *
unit_price * (1 - discount_value)) as total_sales
   from tmp_sales_order_fact
   group by season_id, order_mode_id, product_id, city||'-'||country_id||'-'||region_id, to_char(order_date,
'yyyymm');

-- tmp_employee_fact
create table tmp_employee_fact as
   select e.employee_id, d.department_id, j.job_id,
      l.city || '-' || co.country_id || '-' || co.region_id as city_country_region_id,
      e.salary, e.hire_date, am.year_month_id
   from employees e, departments d, jobs j, locations l, countries co , all_months am
   where e.department_id = d.department_id and e.job_id = j.job_id and l.location_id = d.location_id
   and l.country_id = co.country_id and to_char(e.hire_date, 'yyyymm') <= am.year_month_id;

-- employee_fact
create table employee_fact as
   select department_id, job_id, year_month_id, city_country_region_id, sum(salary) as total_salary,
      count (*) as number_of_employees
   from tmp_employee_fact
   group by department_id, job_id, year_month_id, city_country_region_id;
```

```
-- tmp_customer_fact
create table tmp_customer_fact as
   select cu.credit_limit, cu.city, cu.country_id, co.region_id
   from customers cu, countries co
   where cu.country_id = co.country_id;

alter table tmp_customer_fact add credit_id char(1);

update tmp_customer_fact set credit_id = 'H' where credit_limit > 3500;
update tmp_customer_fact set credit_id = 'M' where credit_limit <= 3500 and credit_limit > 1500;
update tmp_customer_fact set credit_id = 'L' where credit_limit <= 1500;

-- customer_fact
create table customer_fact as
   select credit_id, city || '-' || country_id || '-' || region_id as city_country_region_id,
      count(*) as number_of_customers
   from tmp_customer_fact
   group by credit_id, city || '-' || country_id || '-' || region_id;
```

(b) SQL statements to create the star schemas Version-2.

```
-- order_mode_dim2
create table order_mode_dim2 (order_mode_id char(1), order_mode_desc varchar(20));
insert into order_mode_dim2 values ('O', 'online');
insert into order_mode_dim2 values ('D', 'direct');

-- season_dim2
create table season_dim2 (season_id int, season_desc varchar(10));
insert into season_dim2 values (1, 'Spring');
insert into season_dim2 values (2, 'Summer');
insert into season_dim2 values (3, 'Autumn');
insert into season_dim2 values (4, 'Winter');

-- product_dim2
create table product_dim2 as
   select distinct p.product_id, p.product_name, p.product_description as product_desc,
      oi.unit_price as price
   from products p, order_items oi
   where oi.product_id(+) = p.product_id;
```

```
-- all_months
-- year
create table year2(year char(4));

insert into year values('2004');
insert into year values('2005');
insert into year values('2006');
insert into year values('2007');
insert into year values('2008');

-- month
create table month2(month char(2));

insert into month values('01');
insert into month values('02');
insert into month values('03');
insert into month values('04');
insert into month values('05');
insert into month values('06');
insert into month values('07');
insert into month values('08');
insert into month values('09');
insert into month values('10');
insert into month values('11');
insert into month values('12');

-- all_months
create table all_months2 as select y.year || m.month as year_month_id, y.year, m.month from month2 m,
year2 y;

-- month_year_dim2
create table year_month_dim2 as select year || month as year_month_id, year, month from
   (select distinct to_char(order_date, 'yyyy') as year, to_char(order_date, 'mm') as month from orders
   union
   select distinct year, month from all_months);

-- city_dim2
create table city_dim2 as
   select city || '-' || country_id as city_country_id, city, country_id from
   (select city, country_id from customers union select city, country_id from locations);

-- country_dim2
create table country_dim2 as select * from countries;
```

```
-- regoin_dim2
create table region_dim2 as select * from regions;

-- credit_dim2
create table credit_dim2 (credit_id char(1), credit_name varchar(50), credit_desc varchar(255));
insert into credit_dim2 values ('L', 'low credit', 'credit <= 1500');
insert into credit_dim2 values ('M', 'medium credit', '1500 < credit <= 3500');
insert into credit_dim2 values ('H', 'high credit', 'credit > 3500');

-- department_dim2
create table department_dim2 as select department_id, department_name from departments;

-- job_dim2
create table job_dim2 as select job_id, job_title from jobs;

-- tmp_sales_order_fact2
create table tmp_sales_order_fact2 as
   select o.order_date, o.order_mode, oi.product_id, cu.city, cu.country_id, oi.unit_price,
      oi.quantity, p.discount
   from orders o, order_items oi, customers cu, promotions p
   where o.order_id = oi.order_id and cu.customer_id = o.customer_id and p.promotion_id =
o.promotion_id;

alter table tmp_sales_order_fact2 add discount_value numeric(3,2);
update tmp_sales_order_fact2 set discount_value = 0 where upper(discount) = 'FULL PRICE';
update tmp_sales_order_fact2 set discount_value = 0.3 where upper(discount) = '30% OFF';
update tmp_sales_order_fact2 set discount_value = 0.2 where upper(discount) = '20% OFF';
update tmp_sales_order_fact2 set discount_value = 0.1 where upper(discount) = '10% OFF';

alter table tmp_sales_order_fact2 add season_id int;
update tmp_sales_order_fact2 set season_id = 1 where to_char(order_date, 'mm') >= '09' and
to_char(order_date, 'mm') <= '11';
update tmp_sales_order_fact2 set season_id = 2 where to_char(order_date, 'mm') >= '12' or
to_char(order_date, 'mm') <= '02';
update tmp_sales_order_fact2 set season_id = 3 where to_char(order_date, 'mm') >= '03' and
to_char(order_date, 'mm') <= '05';
update tmp_sales_order_fact2 set season_id = 4 where to_char(order_date, 'mm') >= '06' and
to_char(order_date, 'mm') <= '08';

alter table tmp_sales_order_fact2 add order_mode_id char(1);
update tmp_sales_order_fact2 set order_mode_id = 'O' where upper(order_mode) = 'ONLINE';
update tmp_sales_order_fact2 set order_mode_id = 'D' where upper(order_mode) = 'DIRECT';

-- sales_order_fact2
```

```
create table sales_order_fact2 as
   select season_id, order_mode_id, product_id, city || '-' || country_id as city_country_id,
to_char(order_date, 'yyyymm') as year_month_id,
   count(*) as number_of_orders, sum(unit_price * quantity * (1 - discount_value)) as total_sales
   from tmp_sales_order_fact2
   group by season_id, order_mode_id, product_id, city || '-' || country_id, to_char(order_date, 'yyyymm');

-- tmp_employee_fact2
create table tmp_employee_fact2 as
   select e.employee_id, d.department_id, j.job_id, l.city || '-' || l.country_id as city_country_id,
            e.salary, e.hire_date, am.year_month_id
   from employees e, departments d, jobs j, locations l, countries co , all_months am
   where e.department_id = d.department_id and e.job_id = j.job_id and l.location_id = d.location_id
   and to_char(e.hire_date, 'yyyymm') <= am.year_month_id;

-- employee_fact2
create table employee_fact2 as
   select department_id, job_id, year_month_id, city_country_id, sum(salary) as total_salary, count (*) as
number_of_employees
   from tmp_employee_fact2
   group by department_id, job_id, year_month_id, city_country_id;

-- tmp_customer_fact2
create table tmp_customer_fact2 as
   select credit_limit, country_id from customers;

alter table tmp_customer_fact2 add credit_id char(1);

update tmp_customer_fact2 set credit_id = 'H' where credit_limit > 3500;
update tmp_customer_fact2 set credit_id = 'M' where credit_limit <= 3500 and credit_limit > 1500;
update tmp_customer_fact2 set credit_id = 'L' where credit_limit <= 1500;

-- customer_fact2
create table customer_fact2 as
   select credit_id, country_id, count(*) as number_of_customers
   from tmp_customer_fact2
   group by credit_id, country_id;
```

(c) Screen shots of the tables that you have created

All tables

Star schemas Version-1:

order_mode_dim:

	ORDER_MODE_ID	ORDER_MODE_DESC
1	O	online
2	D	direct

season_dim:

	SEASON_ID	SEASON_DESC
1	1	Spring
2	2	Summer
3	3	Autumn
4	4	Winter

product_dim:

	PRODUCT_ID	PRODUCT_NAME	PRODUCT_DESC	WEIGHT_FACTOR	WAREHOUSE_GROUP_LIST
1	2400	DIMM - 512 MB	512 MB DIMM memor...	1	UNKNOWN
2	3000	Laptop 32/10/56	Envoy Laptop, 32M...	0.11111111111...	1_2_3_4_5_6_7_8_9
3	3300	Screws <S.32.P>	Screws: Steel, si...	0.11111111111...	1_2_3_4_5_6_7_8_9
4	3400	HD 8GB /SE	8GB capacity SCSI...	0.2	2_4_6_8_9
5	1726	LCD Monitor 11/PM	Liquid Cristal Di...	1	UNKNOWN
6	1729	Chemicals - RCP	Cleaning Chemical...	0.16666666666...	3_5_6_7_8_9
7	1733	PS 220V /UK	220V Power supply...	0.11111111111...	1_2_3_4_5_6_7_8_9
8	1734	Cable RS232 10/AM	10 ft RS232 cable...	0.11111111111...	1_2_3_4_5_6_7_8_9
9	1737	Cable SCSI 10/FW/ADS	10ft SCSI2 F/W Ad...	0.11111111111...	1_2_3_4_5_6_7_8_9
10	1738	PS 110V /US	110 V Power Suppl...	0.11111111111...	1_2_3_4_5_6_7_8_9
11	1739	SDRAM - 128 MB	SDRAM memory, 128...	0.2	2_4_6_8_9

	PRODUCT_ID	PRODUCT_NAME	PRODUCT_DESC	WEIGHT_FACTOR	WAREHOUSE_GROUP_LIST
277	3355	HD 8GB /SI	8GB SCSI hard dis...	0.2	2_4_6_8_9
278	3359	SDRAM - 16 MB	SDRAM memory upgr...	0.2	2_4_6_8_9
279	3361	Spreadsheet - SSP/S 1.5	SmartSpread Sprea...	0.33333333333...	2_5_6
280	3362	Web Browser - SB/S 4.0	Software Kit Web ...	0.2	2_5_6_7_8
281	3391	PS 110/220	Power Supply - sw...	0.11111111111...	1_2_3_4_5_6_7_8_9
282	3399	HD 18GB /SE	18GB SCSI externa...	1	UNKNOWN
283	3501	C for SPNIX4.0 - Sys	C programming sof...	0.2	2_5_6_7_8
284	3502	C for SPNIX3.3 -Sys/U	C programming sof...	0.2	2_5_6_7_8
285	3503	C for SPNIX3.3 - Seat/U	C programming sof...	0.2	2_5_6_7_8
286	3511	Paper - HQ Printer	Quality paper for...	1	6
287	3515	Lead Replacement	Refill leads for ...	1	6

bridge_table:

	PRODUCT_ID	WAREHOUSE_ID
1	1733	1
2	2377	1
3	2380	1
4	2418	1
5	3000	1
6	3139	1
7	3300	1
8	1742	2
9	1750	2
10	1769	2
11	1787	2
12	1820	2
13	2264	2

	PRODUCT_ID	WAREHOUSE_ID
1100	2260	9
1101	2319	9
1102	2322	9
1103	2340	9
1104	2374	9
1105	2395	9
1106	2403	9
1107	2419	9
1108	2457	9
1109	2878	9
1110	3090	9
1111	3355	9
1112	3391	9

warehouse_dim:

	WAREHOUSE_ID	WAREHOUSE_SPEC	WAREHOUSE_NAME	LOCATION_ID
1	1	(null)	Southlake, Texas	1400
2	2	(null)	San Francisco	1500
3	3	(null)	New Jersey	1600
4	4	(null)	Seattle, Washington	1700
5	5	(null)	Toronto	1800
6	6	(null)	Sydney	2200
7	7	(null)	Mexico City	3200
8	8	(null)	Beijing	2000
9	9	(null)	Bombay	2100

tmp_promotion:

	DISCOUNT	START_...	END_DATE	DISCOUNT_VALUE
1	30% off	01/DEC/07	31/DEC/07	0.3
2	20% off	01/JUN/07	30/JUN/07	0.2
3	20% off	01/JUN/08	30/JUN/08	0.2
4	10% off	10/NOV/07	11/NOV/07	0.1
5	Full Price	01/JAN/04	31/MAY/07	0
6	Full Price	01/JUL/07	09/NOV/07	0
7	Full Price	12/NOV/07	30/NOV/07	0
8	Full Price	01/JAN/08	31/MAY/08	0
9	Full Price	01/JUL/08	31/DEC/08	0

product_price_dim:

	PRODUCT_ID	START_DATE	END_DATE	DISCOUNT	PRICE_AFTER_DISCOUNT
1	2359	01/DEC/07	31/DEC/07	30% off	173.6
2	2274	01/DEC/07	31/DEC/07	30% off	109.9
3	2403	01/DEC/07	31/DEC/07	30% off	79.31
4	2409	01/DEC/07	31/DEC/07	30% off	136.29
5	2536	01/DEC/07	31/DEC/07	30% off	56
6	2995	01/DEC/07	31/DEC/07	30% off	48.3
7	3110	01/DEC/07	31/DEC/07	30% off	31.5
8	2276	01/DEC/07	31/DEC/07	30% off	165.55
9	3123	01/DEC/07	31/DEC/07	30% off	50.05
10	3086	01/DEC/07	31/DEC/07	30% off	145.6
11	3216	01/DEC/07	31/DEC/07	30% off	21
12	2439	01/DEC/07	31/DEC/07	30% off	121.8
13	2410	01/DEC/07	31/DEC/07	30% off	245.63

	PRODUCT_ID	START_DATE	END_DATE	DISCOUNT	PRICE_AFTER_DISCOUNT
1653	3501	01/JUL/08	31/DEC/08	Full Price	492.8
1654	2402	01/JUL/08	31/DEC/08	Full Price	127
1655	2299	01/JUL/08	31/DEC/08	Full Price	76
1656	3155	01/JUL/08	31/DEC/08	Full Price	43
1657	2412	01/JUL/08	31/DEC/08	Full Price	95
1658	3163	01/JUL/08	31/DEC/08	Full Price	32
1659	3515	01/JUL/08	31/DEC/08	Full Price	1.1
1660	2382	01/JUL/08	31/DEC/08	Full Price	804.1
1661	2319	01/JUL/08	31/DEC/08	Full Price	25
1662	2470	01/JUL/08	31/DEC/08	Full Price	76
1663	2406	01/JUL/08	31/DEC/08	Full Price	195.8
1664	2335	01/JUL/08	31/DEC/08	Full Price	97
1665	2419	01/JUL/08	31/DEC/08	Full Price	69

year:

	YEAR
1	2004
2	2005
3	2006
4	2007
5	2008
6	2004
7	2005
8	2006
9	2007
10	2008

month:

	MONTH
1	01
2	02
3	03
4	04
5	05
6	06
7	07
8	08
9	09
10	10
11	11
12	12
13	01
14	02
15	03
16	04
17	05
18	06
19	07
20	08
21	09
22	10
23	11
24	12

all_months:

	YEAR_MONTH_ID	YEAR	MONTH
1	200401	2004	01
2	200402	2004	02
3	200403	2004	03
4	200404	2004	04
5	200405	2004	05
6	200406	2004	06
7	200407	2004	07
8	200408	2004	08
9	200409	2004	09
10	200410	2004	10

	YEAR_MONTH_ID	YEAR	MONTH
51	200803	2008	03
52	200804	2008	04
53	200805	2008	05
54	200806	2008	06
55	200807	2008	07
56	200808	2008	08
57	200809	2008	09
58	200810	2008	10
59	200811	2008	11
60	200812	2008	12

year_month_dim:

	YEAR_MONTH_ID	YEAR	MONTH
1	200401	2004	01
2	200402	2004	02
3	200403	2004	03
4	200404	2004	04
5	200405	2004	05
6	200406	2004	06
7	200407	2004	07
8	200408	2004	08
9	200409	2004	09
10	200410	2004	10

	YEAR_MONTH_ID	YEAR	MONTH
52	200804	2008	04
53	200805	2008	05
54	200806	2008	06
55	200807	2008	07
56	200808	2008	08
57	200809	2008	09
58	200810	2008	10
59	200811	2008	11
60	200812	2008	12
61	200902	2009	02

city_country_region_dim:

	CITY_COUNTRY_REGION_ID	CITY	COUNTRY_NAME	REGION_NAME
1	Albany-US-2	Albany	United States of America	Americas
2	Alexandria-US-2	Alexandria	United States of America	Americas
3	Altoona-US-2	Altoona	United States of America	Americas
4	Ann Arbor-US-2	Ann Arbor	United States of America	Americas
5	Baden-Daettwil-CH-1	Baden-Daettwil	Switzerland	Europe
6	Baltimore-US-2	Baltimore	United States of America	Americas
7	Bangalore-IN-3	Bangalore	India	Asia
8	Bangalore - India-IN-3	Bangalore - India	India	Asia
9	Batavia-IN-3	Batavia	India	Asia
10	Battle Creek-US-2	Battle Creek	United States of America	Americas

	CITY_COUNTRY_REGION_ID	CITY	COUNTRY_NAME	REGION_NAME
109	Venice-IT-1	Venice	Italy	Europe
110	Ventimiglia-IT-1	Ventimiglia	Italy	Europe
111	Warren-US-2	Warren	United States of America	Americas
112	Wausau-US-2	Wausau	United States of America	Americas
113	White Plains-US-2	White Plains	United States of America	Americas
114	Whitehorse-CA-2	Whitehorse	Canada	Americas
115	Yonkers-US-2	Yonkers	United States of America	Americas
116	Zuerich-CH-1	Zuerich	Switzerland	Europe
117	Zurich-CH-1	Zurich	Switzerland	Europe
118	-IN-3	(null)	India	Asia

credit_dim:

	CREDIT_ID	CREDIT_NAME	CREDIT_DESC
1	L	low credit	credit <= 1500
2	M	medium credit	1500 < credit <= 3500
3	H	high credit	credit > 3500

department_dim:

	DEPARTMENT_ID	DEPARTMENT_NAME
1	10	Administration
2	20	Marketing
3	30	Purchasing
4	40	Human Resources
5	50	Shipping
6	60	IT
7	70	Public Relations
8	80	Sales
9	90	Executive
10	100	Finance

	DEPARTMENT_ID	DEPARTMENT_NAME
18	180	Construction
19	190	Contracting
20	200	Operations
21	210	IT Support
22	220	NOC
23	230	IT Helpdesk
24	240	Government Sales
25	250	Retail Sales
26	260	Recruiting
27	270	Payroll

job_dim:

	JOB_ID	JOB_TITLE
1	AD_PRES	President
2	AD_VP	Administration Vice President
3	AD_ASST	Administration Assistant
4	FI_MGR	Finance Manager
5	FI_ACCOUNT	Accountant
6	AC_MGR	Accounting Manager
7	AC_ACCOUNT	Public Accountant
8	SA_MAN	Sales Manager
9	SA_REP	Sales Representative
10	PU_MAN	Purchasing Manager
11	PU_CLERK	Purchasing Clerk
12	ST_MAN	Stock Manager
13	ST_CLERK	Stock Clerk
14	SH_CLERK	Shipping Clerk
15	IT_PROG	Programmer
16	MK_MAN	Marketing Manager
17	MK_REP	Marketing Representative
18	HR_REP	Human Resources Representative
19	PR_REP	Public Relations Representative

tmp_sales_order_fact:

	ORDER_DATE	ORDER_MODE	PRODUCT_ID	CITY	COUNTRY_ID	REGION_ID	UNIT_PRICE	QUANTITY	DISCOUNT	DISCOUNT_VALUE	SEASON_ID	ORDER_MODE_ID
1	26/JAN/06 10:22:51.962632000 AM	online	2289	Indianapolis	US	2	48	200	Full Price	0	2	O
2	26/JAN/08 10:22:41.934562000 AM	online	2264	Bloomington	US	2	199.1	38	Full Price	0	2	O
3	08/JAN/06 09:19:44.123456000 PM	direct	2211	Indianapolis	US	2	3.3	140	Full Price	0	2	D
4	08/JAN/08 06:03:12.654278000 PM	direct	1781	Bloomington	US	2	226.6	9	Full Price	0	2	D
5	01/JAN/08 06:03:12.654278000 PM	direct	2337	Indianapolis	US	2	270.6	1	Full Price	0	2	D
6	14/NOV/07 01:22:31.223344000 PM	online	2058	Elkhart	US	2	23	29	Full Price	0	1	O
7	13/NOV/07 02:34:21.986210000 PM	online	2289	Indianapolis	US	2	48	180	Full Price	0	1	O
8	13/NOV/07 03:41:10.619477000 PM	online	2289	South Bend	US	2	48	200	Full Price	0	1	O
9	23/OCT/07 04:49:56.346122000 PM	online	2264	Cedar Rapids	US	2	199.1	9	Full Price	0	1	O
10	28/AUG/07 05:18:45.942399000 PM	online	1910	Eau Claire	US	2	14	6	Full Price	0	4	O
11	28/AUG/07 06:03:34.003399000 PM	online	2289	Milwaukee	US	2	48	92	Full Price	0	4	O

	ORDER_DATE	ORDER_MODE	PRODUCT_ID	CITY	COUNTRY_ID	REGION_ID	UNIT_PRICE	QUANTITY	DISCOUNT	DISCOUNT_VALUE	SEASON_ID	ORDER_MODE_ID
656	27/JUL/08 07:59:10.223344000 AM	direct	2311	Kokomo	US	2	93	44	Full Price	0	4	D
657	06/OCT/07 07:59:43.462632000 PM	direct	3173	Madison	US	2	80	23	Full Price	0	1	D
658	20/SEP/07 10:34:11.456789000 AM	direct	2536	Eau Claire	US	2	80	54	Full Price	0	1	D
659	31/OCT/07 10:22:16.162632000 PM	direct	3172	Grand Rapids	US	2	37	45	Full Price	0	1	D
660	16/AUG/07 02:34:12.234359000 PM	direct	3163	Kokomo	US	2	32	142	Full Price	0	4	D
661	26/JAN/06 10:22:51.962632000 AM	online	2359	Indianapolis	US	2	248	204	Full Price	0	2	O
662	08/JAN/06 09:19:44.123456000 PM	direct	2289	Indianapolis	US	2	48	41	Full Price	0	2	D
663	01/JAN/08 06:03:12.654278000 PM	direct	2381	Indianapolis	US	2	97	17	Full Price	0	2	D
664	13/NOV/07 02:34:21.986210000 PM	online	2365	Indianapolis	US	2	77	209	Full Price	0	1	O
665	18/AUG/07 02:34:12.234359000 PM	direct	2350	South Bend	US	2	2341.9	6	20% off	0.2	4	D
666	08/JAN/08 06:03:12.654278000 PM	direct	1808	Bloomington	US	2	55	14	Full Price	0	2	D

sales_order_fact:

	SEASON_ID	ORDER_MODE_ID	PRODUCT_ID	CITY_COUNTRY_REGION_ID	YEAR_MONTH_ID	NUMBER_OF_ORDERS	TOTAL_SALES
1	2	O	2264	Bloomington-US-2	200801	1	7565.8
2	1	O	2058	Elkhart-US-2	200711	1	667
3	1	O	2264	Cedar Rapids-US-2	200710	1	1791.9
4	4	O	3150	Detroit-US-2	200706	1	40.8
5	3	O	3106	Des Moines-US-2	200705	1	1248
6	3	O	2409	Eau Claire-US-2	200805	1	7203.9
7	1	D	2471	Milwaukee-US-2	200711	1	2414.5
8	3	D	2870	Indianapolis-US-2	200703	1	44
9	3	D	3106	Indianapolis-US-2	200703	1	5280
10	1	D	3220	Eau Claire-US-2	200711	1	328
11	1	D	2430	Madison-US-2	200711	1	1879.2

	SEASON_ID	ORDER_MODE_ID	PRODUCT_ID	CITY_COUNTRY_REGION_ID	YEAR_MONTH_ID	NUMBER_OF_ORDERS	TOTAL_SALES
656	3	D	3163	Kokomo-US-2	200803	1	2112
657	1	D	3248	Madison-US-2	200611	1	5519.8
658	2	O	3170	Saginaw-US-2	200702	1	5227.2
659	2	O	3176	Lansing-US-2	200702	1	12349.7
660	2	O	2350	Milwaukee-US-2	200712	1	178686.97
661	2	O	2394	Madison-US-2	200712	1	2938.32
662	4	D	2782	Madison-US-2	200706	1	1537.6
663	1	D	3165	Grand Rapids-US-2	200711	1	2170.8
664	1	D	2496	Bloomington-US-2	200609	1	9394
665	2	D	2381	Indianapolis-US-2	200801	1	1649
666	2	D	1808	Bloomington-US-2	200801	1	770

tmp_employee_fact:

	EMPLOYEE_ID	DEPARTMENT_ID	JOB_ID	CITY_COUNTRY_REGION_ID	SALARY	HIRE_DATE	YEAR_MONTH_ID
1	102	90	AD_VP	Seattle-US-2	17000	13/JAN/01	200401
2	102	90	AD_VP	Seattle-US-2	17000	13/JAN/01	200402
3	102	90	AD_VP	Seattle-US-2	17000	13/JAN/01	200403
4	102	90	AD_VP	Seattle-US-2	17000	13/JAN/01	200404
5	102	90	AD_VP	Seattle-US-2	17000	13/JAN/01	200405
6	102	90	AD_VP	Seattle-US-2	17000	13/JAN/01	200406
7	102	90	AD_VP	Seattle-US-2	17000	13/JAN/01	200407
8	102	90	AD_VP	Seattle-US-2	17000	13/JAN/01	200408
9	102	90	AD_VP	Seattle-US-2	17000	13/JAN/01	200409
10	102	90	AD_VP	Seattle-US-2	17000	13/JAN/01	200410
11	102	90	AD_VP	Seattle-US-2	17000	13/JAN/01	200411

	EMPLOYEE_ID	DEPARTMENT_ID	JOB_ID	CITY_COUNTRY_REGION_ID	SALARY	HIRE_DATE	YEAR_MONTH_ID
3846	173	80	SA_REP	Oxford-UK-1	6100	21/APR/08	200811
3847	173	80	SA_REP	Oxford-UK-1	6100	21/APR/08	200812
3848	167	80	SA_REP	Oxford-UK-1	6200	21/APR/08	200804
3849	167	80	SA_REP	Oxford-UK-1	6200	21/APR/08	200805
3850	167	80	SA_REP	Oxford-UK-1	6200	21/APR/08	200806
3851	167	80	SA_REP	Oxford-UK-1	6200	21/APR/08	200807
3852	167	80	SA_REP	Oxford-UK-1	6200	21/APR/08	200808
3853	167	80	SA_REP	Oxford-UK-1	6200	21/APR/08	200809
3854	167	80	SA_REP	Oxford-UK-1	6200	21/APR/08	200810
3855	167	80	SA_REP	Oxford-UK-1	6200	21/APR/08	200811
3856	167	80	SA_REP	Oxford-UK-1	6200	21/APR/08	200812

employee_fact:

	DEPARTMENT_ID	JOB_ID	YEAR_MONTH_ID	CITY_COUNTRY_REGION_ID	TOTAL_SALARY	NUMBER_OF_EMPLOYEES
1	90	AD_VP	200405	Seattle-US-2	17000	1
2	90	AD_VP	200503	Seattle-US-2	17000	1
3	90	AD_VP	200712	Seattle-US-2	34000	2
4	110	AC_ACCOUNT	200401	Seattle-US-2	9300	1
5	110	AC_ACCOUNT	200405	Seattle-US-2	9300	1
6	110	AC_ACCOUNT	200506	Seattle-US-2	9300	1
7	110	AC_ACCOUNT	200607	Seattle-US-2	9300	1
8	110	AC_ACCOUNT	200611	Seattle-US-2	9300	1
9	110	AC_ACCOUNT	200612	Seattle-US-2	9300	1
10	110	AC_ACCOUNT	200701	Seattle-US-2	9300	1
11	110	AC_ACCOUNT	200802	Seattle-US-2	9300	1

	DEPARTMENT_ID	JOB_ID	YEAR_MONTH_ID	CITY_COUNTRY_REGION_ID	TOTAL_SALARY	NUMBER_OF_EMPLOYEES
1084	20	MK_REP	200509	Toronto-CA-2	6000	1
1085	20	MK_REP	200601	Toronto-CA-2	6000	1
1086	20	MK_REP	200602	Toronto-CA-2	6000	1
1087	20	MK_REP	200604	Toronto-CA-2	6000	1
1088	20	MK_REP	200605	Toronto-CA-2	6000	1
1089	20	MK_REP	200606	Toronto-CA-2	6000	1
1090	20	MK_REP	200610	Toronto-CA-2	6000	1
1091	20	MK_REP	200611	Toronto-CA-2	6000	1
1092	20	MK_REP	200702	Toronto-CA-2	6000	1
1093	20	MK_REP	200708	Toronto-CA-2	6000	1
1094	20	MK_REP	200807	Toronto-CA-2	6000	1

tmp_customer_fact:

	CREDIT_LIMIT	CITY	COUNTRY_ID	REGION_ID	CREDIT_ID
1	2400	Erie	US	2	M
2	1500	Chennai	IN	3	L
3	500	Roma	IT	1	L
4	500	Roma	IT	1	L
5	2400	Roma	IT	1	M
6	700	Ventimiglia	IT	1	L
7	700	Roma	IT	1	L
8	900	Milwaukee	US	2	L
9	1200	Minneapolis	US	2	L
10	1200	Albany	US	2	L
11	1200	Schenectady	US	2	L

	CREDIT_LIMIT	CITY	COUNTRY_ID	REGION_ID	CREDIT_ID
310	2400	Laurel	US	2	M
311	3600	Milwaukee	US	2	H
312	3600	Buffalo	US	2	H
313	3700	Detroit	US	2	H
314	1200	Muang Chonburi	CN	3	L
315	100	Elkhart	US	2	L
316	200	Grand Rapids	US	2	L
317	300	Ann Arbor	US	2	L
318	400	Kalamazoo	US	2	L
319	700	Milwaukee	US	2	L
320	1900	Philadelphia	US	2	M

customer_fact:

	CREDIT_ID	CITY_COUNTRY_REGION_ID	NUMBER_OF_CUSTOMERS
1	L	Roma-IT-1	7
2	M	Baden-Daettwil-CH-1	8
3	H	Bangalore-IN-3	7
4	M	Ocean City-US-2	2
5	L	Firenze-IT-1	4
6	H	Pontiac-US-2	1
7	L	Toronto-CA-2	1
8	L	Reading-US-2	4
9	L	Cochin-IN-3	2
10	M	Council Bluffs-US-2	2
11	H	Rochester-US-2	1

	CREDIT_ID	CITY_COUNTRY_REGION_ID	NUMBER_OF_CUSTOMERS
142	L	New Delhi - India-IN-3	1
143	L	Kokomo-US-2	1
144	M	Grand Rapids-US-2	2
145	M	Munich-DE-1	3
146	L	Ann Arbor-US-2	4
147	M	Scranton-US-2	3
148	M	Chennai-IN-3	1
149	H	Cumberland-US-2	1
150	L	La Crosse-US-2	1
151	H	Peking-CN-3	1
152	H	Baden-Daettwil-CH-1	1

Star schemas Version-2:

order_mode_dim2:

	ORDER_MODE_ID	ORDER_MODE_DESC
1	O	online
2	D	direct

season_dim2:

	SEASON_ID	SEASON_DESC
1	1	Spring
2	2	Summer
3	3	Autumn
4	4	Winter

product_dim2:

	PRODUCT_ID	PRODUCT_NAME	PRODUCT_DESC	PRICE
1	1788	CPU D600	Dual CPU @ 600Mhz. State of the art, hig...	(null)
2	3133	Video Card /32	Video Card, with 32MB cache memory.	43
3	2497	WSP DA-290	Wide storage processor (model DA-290).	(null)
4	3110	KB 101/FR	Standard PC/AT Enhanced Keyboard (101/10...	45
5	3108	KB E/EN	Ergonomic Keyboard with two separate key...	77
6	2944	Wrist Pad /CL	Wrist Pad with corporate logo	(null)
7	2403	CD-ROM 600/I/24x	600 MB internal read only CD-ROM drive, ...	113.3
8	2270	Modem - 56/90/I	Modem - 56kb per second, v.90 PCI Global...	66
9	3099	Cable Harness	Cable harness to organize and bundle com...	3.3
10	3391	PS 110/220	Power Supply - switchable, 110V/220V	(null)
11	2370	PS 220V /HS/FR	220V hot swappable power supply, for Fra...	91
12	1743	HD 18.2GB @10000 /E	External hard drive disk - 18.2 GB, rate...	(null)
13	3090	RAM - 48 MB	Random Access Memory, SIMM - 48 MB capac...	187
14	1739	SDRAM - 128 MB	SDRAM memory. 128 MB capacity. SDRAM can...	(null)

	PRODUCT_ID	PRODUCT_NAME	PRODUCT_DESC	PRICE
274	3183	Word Processing - SWS/...	SmartWord Word Processor, Standard Editi...	47
275	2470	SPNIX4.0 - NL	Operating System Software: SPNIX V4.0 - ...	76
276	3503	C for SPNIX3.3 - Seat/U	C programming software for SPNIX V3.3 - ...	(null)
277	1825	X25 - 1 Line License	X25 network access control system, singl...	24
278	3362	Web Browser - SB/S 4.0	Software Kit Web Browser: SmartBrowse V4...	94
279	2335	Mobile phone	Dual band mobile phone with low battery ...	97
280	2350	Desk - W/48	Desk - 48 inch white laminate without re...	2341.9
281	2091	Paper Tablet LW 8 1/2 ...	Paper tablet, lined, white, size 8 1/2 x...	(null)
282	2336	Business Cards Box - 250	Business cards box, capacity 250. Use fo...	(null)
283	2339	Paper - Std Printer	20 lb. 8.5x11 inch white laser printer p...	25
284	2810	Inkvisible Pens	Rollerball pen is equipped with a smooth...	6
285	3209	Sharpener - Pencil	Electric Pencil Sharpener Rugged steel c...	13
286	3511	Paper - HQ Printer	Quality paper for inkjet and laser print...	9
287	2334	Resin	General purpose synthetic resin.	3.3

year2:

	YEAR
1	2004
2	2005
3	2006
4	2007
5	2008
6	2004
7	2005
8	2006
9	2007
10	2008

month2:

	MONTH
1	01
2	02
3	03
4	04
5	05
6	06
7	07
8	08
9	09
10	10
11	11
12	12
13	01
14	02
15	03
16	04
17	05
18	06
19	07
20	08
21	09
22	10
23	11
24	12

all_months2:

	YEAR_MONTH_ID	YEAR	MONTH
1	200401	2004	01
2	200402	2004	02
3	200403	2004	03
4	200404	2004	04
5	200405	2004	05
6	200406	2004	06
7	200407	2004	07
8	200408	2004	08
9	200409	2004	09
10	200410	2004	10

	YEAR_MONTH_ID	YEAR	MONTH
51	200803	2008	03
52	200804	2008	04
53	200805	2008	05
54	200806	2008	06
55	200807	2008	07
56	200808	2008	08
57	200809	2008	09
58	200810	2008	10
59	200811	2008	11
60	200812	2008	12

year_month_dim2:

	YEAR_MONTH_ID	YEAR	MONTH
1	200401	2004	01
2	200402	2004	02
3	200403	2004	03
4	200404	2004	04
5	200405	2004	05
6	200406	2004	06
7	200407	2004	07
8	200408	2004	08
9	200409	2004	09
10	200410	2004	10

	YEAR_MONTH_ID	YEAR	MONTH
52	200804	2008	04
53	200805	2008	05
54	200806	2008	06
55	200807	2008	07
56	200808	2008	08
57	200809	2008	09
58	200810	2008	10
59	200811	2008	11
60	200812	2008	12
61	200902	2009	02

city_dim2:

	CITY_COUNTRY_ID	CITY	COUNTRY_ID
1	Albany-US	Albany	US
2	Alexandria-US	Alexandria	US
3	Altoona-US	Altoona	US
4	Ann Arbor-US	Ann Arbor	US
5	Baden-Daettwil-CH	Baden-Daettwil	CH
6	Baltimore-US	Baltimore	US
7	Bangalore-IN	Bangalore	IN
8	Bangalore - India-IN	Bangalore - India	IN
9	Batavia-IN	Batavia	IN
10	Battle Creek-US	Battle Creek	US
11	Beijing-CN	Beijing	CN
12	Bel Air-US	Bel Air	US
13	Beloit-US	Beloit	US

	CITY_COUNTRY_ID	CITY	COUNTRY_ID
106	Trogen-CH	Trogen	CH
107	Tschiertschen-CH	Tschiertschen	CH
108	Utrecht-NL	Utrecht	NL
109	Venice-IT	Venice	IT
110	Ventimiglia-IT	Ventimiglia	IT
111	Warren-US	Warren	US
112	Wausau-US	Wausau	US
113	White Plains-US	White Plains	US
114	Whitehorse-CA	Whitehorse	CA
115	Yonkers-US	Yonkers	US
116	Zuerich-CH	Zuerich	CH
117	Zurich-CH	Zurich	CH
118	-IN	(null)	IN

country_dim2:

	COUNTRY_ID	COUNTRY_NAME	REGION_ID
1	CA	Canada	2
2	EG	Egypt	4
3	MX	Mexico	2
4	TH	Thailand	3
5	US	United States of America	2
6	KW	Kuwait	4
7	NG	Nigeria	4
8	NL	Netherlands	1
9	JP	Japan	3
10	ML	Malaysia	3
11	UK	United Kingdom	1
12	AR	Argentina	2
13	CH	Switzerland	1

	COUNTRY_ID	COUNTRY_NAME	REGION_ID
15	IN	India	3
16	SG	Singapore	3
17	ZM	Zambia	4
18	BE	Belgium	1
19	CN	China	3
20	DK	Denmark	1
21	FR	France	1
22	IL	Israel	4
23	AU	Australia	3
24	BR	Brazil	2
25	IT	Italy	1
26	ZW	Zimbabwe	4

region_dim2:

	REGION_ID	REGION_NAME
1	1	Europe
2	2	Americas
3	3	Asia
4	4	Middle East and Africa

credit_dim2:

	CREDIT_ID	CREDIT_NAME	CREDIT_DESC
1	L	low credit	credit <= 1500
2	M	medium credit	1500 < credit <= 3500
3	H	high credit	credit > 3500

department_dim2:

	DEPARTMENT_ID	DEPARTMENT_NAME
1	10	Administration
2	20	Marketing
3	30	Purchasing
4	40	Human Resources
5	50	Shipping
6	60	IT
7	70	Public Relations
8	80	Sales
9	90	Executive
10	100	Finance

	DEPARTMENT_ID	DEPARTMENT_NAME
18	180	Construction
19	190	Contracting
20	200	Operations
21	210	IT Support
22	220	NOC
23	230	IT Helpdesk
24	240	Government Sales
25	250	Retail Sales
26	260	Recruiting
27	270	Payroll

job_dim2:

	JOB_ID	JOB_TITLE
1	AD_PRES	President
2	AD_VP	Administration Vice President
3	AD_ASST	Administration Assistant
4	FI_MGR	Finance Manager
5	FI_ACCOUNT	Accountant
6	AC_MGR	Accounting Manager
7	AC_ACCOUNT	Public Accountant
8	SA_MAN	Sales Manager
9	SA_REP	Sales Representative
10	PU_MAN	Purchasing Manager
11	PU_CLERK	Purchasing Clerk
12	ST_MAN	Stock Manager
13	ST_CLERK	Stock Clerk
14	SH_CLERK	Shipping Clerk
15	IT_PROG	Programmer
16	MK_MAN	Marketing Manager
17	MK_REP	Marketing Representative
18	HR_REP	Human Resources Representative
19	PR_REP	Public Relations Representative

tmp_sales_order_fact:

	ORDER_DATE	ORDER_MODE	PRODUCT_ID	CITY	COUNTRY_ID	UNIT_PRICE	QUANTITY	DISCOUNT	DISCOUNT_VALUE	SEASON_ID	ORDER_MODE_ID
1	26/JAN/06 10:22:51.962632000 AM	online	2289	Indianapolis	US	48	200	Full Price	0	2	0
2	26/JAN/08 10:22:41.934562000 AM	online	2264	Bloomington	US	199.1	38	Full Price	0	2	0
3	08/JAN/06 09:19:44.123456000 PM	direct	2211	Indianapolis	US	3.3	140	Full Price	0	2	D
4	08/JAN/08 06:03:12.654278000 PM	direct	1781	Bloomington	US	226.6	9	Full Price	0	2	D
5	01/JAN/08 06:03:12.654278000 PM	direct	2337	Indianapolis	US	270.6	1	Full Price	0	2	D
6	14/NOV/07 01:22:31.223344000 PM	online	2058	Elkhart	US	23	29	Full Price	0	1	0
7	13/NOV/07 02:34:21.986210000 PM	online	2289	Indianapolis	US	48	180	Full Price	0	1	0
8	13/NOV/07 03:41:10.619477000 PM	online	2289	South Bend	US	48	200	Full Price	0	1	0
9	23/OCT/07 04:49:56.346122000 PM	online	2264	Cedar Rapids	US	199.1	9	Full Price	0	1	0
10	28/AUG/07 05:18:45.942399000 PM	online	1910	Eau Claire	US	14	6	Full Price	0	4	0
11	28/AUG/07 06:03:34.003399000 PM	online	2289	Milwaukee	US	48	92	Full Price	0	4	0
12	28/AUG/07 07:59:23.144778000 PM	online	2359	Milwaukee	US	248	8	Full Price	0	4	0

	ORDER_DATE	ORDER_MODE	PRODUCT_ID	CITY	COUNTRY_ID	UNIT_PRICE	QUANTITY	DISCOUNT	DISCOUNT_VALUE	SEASON_ID	ORDER_MODE_ID
655	31/OCT/07 10:22:16.162632000 PM	direct	3172	Grand Rapids	US	37	45	Full Price	0	1	D
656	16/AUG/07 02:34:12.234359000 PM	direct	3163	Kokomo	US	32	142	Full Price	0	4	D
657	26/JAN/06 10:22:51.962632000 AM	online	2359	Indianapolis	US	248	204	Full Price	0	2	0
658	08/JAN/06 09:19:44.123456000 PM	direct	2289	Indianapolis	US	48	41	Full Price	0	2	D
659	01/JAN/08 06:03:12.654278000 PM	direct	2381	Indianapolis	US	97	17	Full Price	0	2	D
660	13/NOV/07 02:34:21.986210000 PM	online	2365	Indianapolis	US	77	209	Full Price	0	1	0
661	18/AUG/07 02:34:12.234359000 PM	direct	2350	South Bend	US	2341.9	6	20% off	0.2	4	D
662	17/DEC/07 06:03:52.562632000 PM	direct	1910	Madison	US	14	9	30% off	0.3	2	D
663	27/JUL/07 01:22:27.462632000 PM	direct	3133	South Bend	US	43	90	Full Price	0	4	D
664	13/SEP/07 04:49:30.647893000 AM	direct	2236	Madison	US	949.3	84	Full Price	0	1	D
665	29/JUN/07 06:03:21.526005000 AM	direct	2761	Milwaukee	US	26	21	20% off	0.2	4	D
666	02/FEB/06 02:34:56.345678000 AM	direct	3163	Milwaukee	US	32	100	Full Price	0	2	D

sales_order_fact2:

	SEASON_ID	ORDER_MODE_ID	PRODUCT_ID	CITY_COUNTRY_ID	YEAR_MONTH_ID	NUMBER_OF_ORDERS	TOTAL_SALES
1	2	D	1781	Bloomington-US	200801	1	2039.4
2	4	0	1910	Eau Claire-US	200708	1	84
3	4	D	3106	Indianapolis-US	200807	1	2928
4	4	0	3150	Detroit-US	200706	1	40.8
5	2	0	3106	Saginaw-US	200702	1	288
6	3	0	3106	Des Moines-US	200705	1	1248
7	3	0	2409	Eau Claire-US	200805	1	7203.9
8	2	D	2211	Milwaukee-US	200602	1	363
9	4	D	2721	Madison-US	200706	1	340
10	1	D	3220	Eau Claire-US	200711	1	328
11	1	D	2211	Madison-US	200709	1	267.3
12	4	D	3117	South Bend-US	200707	1	4180

	SEASON_ID	ORDER_MODE_ID	PRODUCT_ID	CITY_COUNTRY_ID	YEAR_MONTH_ID	NUMBER_OF_ORDERS	TOTAL_SALES
655	4	D	2330	Grand Rapids-US	200707	1	39.6
656	1	D	3170	Madison-US	200710	1	2904
657	1	D	2594	Milwaukee-US	200711	1	243
658	3	D	3051	Minneapolis-US	200805	1	252
659	3	D	3165	Elkhart-US	200703	1	2736
660	4	D	2350	Elkhart-US	200708	1	56205.6
661	4	D	2308	Kokomo-US	200807	1	2240
662	3	D	3183	South Bend-US	200805	1	2961
663	2	0	2394	Madison-US	200712	1	2938.32
664	3	D	3167	Minneapolis-US	200603	1	5076
665	2	D	3167	Cedar Rapids-US	200712	1	1474.2
666	1	D	3290	Eau Claire-US	200711	1	2145

tmp_employee_fact2:

	EMPLOYEE_ID	DEPARTMENT_ID	JOB_ID	CITY_COUNTRY_ID	SALARY	HIRE_DATE	YEAR_MONTH_ID
1	102	90	AD_VP	Seattle-US	17000	13/JAN/01	200401
2	102	90	AD_VP	Seattle-US	17000	13/JAN/01	200402
3	102	90	AD_VP	Seattle-US	17000	13/JAN/01	200403
4	102	90	AD_VP	Seattle-US	17000	13/JAN/01	200404
5	102	90	AD_VP	Seattle-US	17000	13/JAN/01	200405
6	102	90	AD_VP	Seattle-US	17000	13/JAN/01	200406
7	102	90	AD_VP	Seattle-US	17000	13/JAN/01	200407
8	102	90	AD_VP	Seattle-US	17000	13/JAN/01	200408
9	102	90	AD_VP	Seattle-US	17000	13/JAN/01	200409
10	102	90	AD_VP	Seattle-US	17000	13/JAN/01	200410
11	102	90	AD_VP	Seattle-US	17000	13/JAN/01	200411
12	102	90	AD_VP	Seattle-US	17000	13/JAN/01	200412

	EMPLOYEE_ID	DEPARTMENT_ID	JOB_ID	CITY_COUNTRY_ID	SALARY	HIRE_DATE	YEAR_MONTH_ID
16788	100	90	AD_PRES	Seattle-US	24000	17/JUN/03	200712
16789	100	90	AD_PRES	Seattle-US	24000	17/JUN/03	200801
16790	100	90	AD_PRES	Seattle-US	24000	17/JUN/03	200802
16791	100	90	AD_PRES	Seattle-US	24000	17/JUN/03	200803
16792	100	90	AD_PRES	Seattle-US	24000	17/JUN/03	200804
16793	100	90	AD_PRES	Seattle-US	24000	17/JUN/03	200805
16794	100	90	AD_PRES	Seattle-US	24000	17/JUN/03	200806
16795	100	90	AD_PRES	Seattle-US	24000	17/JUN/03	200807
16796	100	90	AD_PRES	Seattle-US	24000	17/JUN/03	200808
16797	100	90	AD_PRES	Seattle-US	24000	17/JUN/03	200809
16798	100	90	AD_PRES	Seattle-US	24000	17/JUN/03	200810
16799	100	90	AD_PRES	Seattle-US	24000	17/JUN/03	200811

employee_fact2:

	DEPARTMENT_ID	JOB_ID	YEAR_MONTH_ID	CITY_COUNTRY_ID	TOTAL_SALARY	NUMBER_OF_EMPLOYEES
1	90	AD_VP	200505	Seattle-US	442000	26
2	90	AD_VP	200509	Seattle-US	884000	52
3	90	AD_VP	200512	Seattle-US	884000	52
4	90	AD_VP	200605	Seattle-US	884000	52
5	90	AD_VP	200702	Seattle-US	884000	52
6	90	AD_VP	200706	Seattle-US	884000	52
7	90	AD_VP	200708	Seattle-US	884000	52
8	90	AD_VP	200802	Seattle-US	884000	52
9	90	AD_VP	200805	Seattle-US	884000	52
10	70	PR_REP	200407	Munich-DE	260000	26
11	70	PR_REP	200506	Munich-DE	260000	26
12	70	PR_REP	200605	Munich-DE	260000	26

	DEPARTMENT_ID	JOB_ID	YEAR_MONTH_ID	CITY_COUNTRY_ID	TOTAL_SALARY	NUMBER_OF_EMPLOYEES
1083	60	IT_PROG	200607	Southlake-US	483600	78
1084	60	IT_PROG	200610	Southlake-US	483600	78
1085	60	IT_PROG	200611	Southlake-US	483600	78
1086	60	IT_PROG	200702	Southlake-US	592800	104
1087	60	IT_PROG	200706	Southlake-US	748800	130
1088	60	IT_PROG	200812	Southlake-US	748800	130
1089	20	MK_REP	200508	Toronto-CA	156000	26
1090	20	MK_REP	200509	Toronto-CA	156000	26
1091	20	MK_REP	200601	Toronto-CA	156000	26
1092	20	MK_REP	200611	Toronto-CA	156000	26
1093	20	MK_REP	200706	Toronto-CA	156000	26
1094	20	MK_REP	200807	Toronto-CA	156000	26

tmp_customer_fact2:

	CREDIT_LIMIT	COUNTRY_ID	CREDIT_ID
1	2400	US	M
2	1500	IN	L
3	500	IT	L
4	500	IT	L
5	2400	IT	M
6	700	IT	L
7	700	IT	L
8	900	US	L
9	1200	US	L
10	1200	US	L
11	1200	US	L
12	1200	US	L

	CREDIT_LIMIT	COUNTRY_ID	CREDIT_ID
309	2300	IN	M
310	2400	US	M
311	3600	US	H
312	3600	US	H
313	3700	US	H
314	1200	CN	L
315	100	US	L
316	200	US	L
317	300	US	L
318	400	US	L
319	700	US	L
320	1900	US	M

customer_fact2:

	CREDIT_ID	COUNTRY_ID	NUMBER_OF_CUSTOMERS
1	L	US	114
2	H	IN	23
3	H	US	27
4	L	IT	29
5	M	JP	1
6	L	CH	12
7	L	CA	1
8	H	TH	1
9	L	IN	23
10	M	DE	3
11	H	CN	1
12	L	TH	1
13	M	IT	6
14	H	IT	8
15	M	CH	14
16	M	IN	8
17	M	US	42
18	L	CN	3
19	H	CH	3

(d) A comparison between the two versions of the star schemas.

Hierarchy

No fact table in version-1 uses hierarchy, whereas in version-2, the Sales order fact and Employee fact tables use hierarchy. The hierarchy in version-2 is country, region, city dimensions.

Bridge Table

Only version-1 has a bridge table between the Product dim table and Warehouse dim table. Hence, version-1 can drill down the product and find out the details of the Warehouse, but version-2 cannot.

Temporal

In Version-1, the Sales order fact table uses Temporal data warehousing SCD Type 4, but Version-2 uses SCD Type 0. Version-1 creates a new dimension to maintain the history of price change, whereas Version-2 does not record the history of price.

Task4 OLAP queries

a. Reports with proper sub-totals:

REPORT 1

(a) Query.

Calculate the total sales of all order modes and all seasons, the subtotal sales of all order modes in each season, the subtotal sales of all seasons by each order mode and the subtotal sales of the combination of each season and each order mode.

(b) Explanation on why such a query is necessary or useful for the management

Because this query can help the management drill down the lower level of aggregation of the total sales. In this case, the total sales of each order mode in each season will help manager to compare how the season and order mode influence total sales and make a strategy on promotion. For example, if the direct mode in spring generated higher sales than the online mode, so the management can encourage company to conduct direct mode in this specific time.

(c) SQL commands.

```
select decode(grouping(o.order_mode_desc), 1, 'All Order Modes', o.order_mode_desc) as order_mode,
   decode(grouping(s.season_desc), 1, 'All Seasons', s.season_desc) as season,
   nvl(sum(sf.total_sales), 0) as total_sales
from sales_order_fact sf, order_mode_dim o, season_dim s
where sf.season_id(+) = s.season_id and sf.order_mode_id(+) = o.order_mode_id
group by cube(o.order_mode_desc, s.season_desc);
```

(d) The screenshots of the query results.

	ORDER_MODE	SEASON	TOTAL_SALES
1	All Order Modes	All Seasons	3478727.31
2	All Order Modes	Autumn	697656.9
3	All Order Modes	Spring	872398.08
4	All Order Modes	Summer	765232.19
5	All Order Modes	Winter	1143440.14
6	direct	All Seasons	1910487.05
7	direct	Autumn	371278.8
8	direct	Spring	638627.48
9	direct	Summer	285255.79
10	direct	Winter	615324.98
11	online	All Seasons	1568240.26
12	online	Autumn	326378.1
13	online	Spring	233770.6
14	online	Summer	479976.4
15	online	Winter	528115.16

REPORT 2

(a) Query.

Calculate the total salary of each department, the subtotal salary of each department by each job in all years, the subtotal salary of each department with a combination of each job and each year.

(b) Explanation on why such a query is necessary or useful for the management.

It is important to the management to estimate the total cost for the company at the begin of each year. Because different department has different job, the overhead is also different. This query can help the management effectively evaluate and modify the total salary for a specific job in a specific department for a next year.

(c) SQL commands.

```
select d.department_name,
   decode(grouping(j.job_title), 1, 'All Jobs', j.job_title) as job,
   decode(grouping(y.year), 1, 'All years', y.year) as year,
   sum(ef.total_salary) as total_salary
from employee_fact ef, department_dim d, job_dim j, year_month_dim y
where j.job_id = ef.job_id and ef.department_id = d.department_id and y.year_month_id = ef.year_month_id
group by d.department_name, rollup(j.job_title, y.year);
```

(d) The screenshots of the query results.

	DEPARTMENT_NAME	JOB	YEAR	TOTAL_SALARY
1	IT	Programmer	2005	33600
2	IT	Programmer	2006	218400
3	IT	Programmer	2007	317400
4	IT	Programmer	2008	345600
5	IT	Programmer	All years	915000
6	IT	All Jobs	All years	915000
7	Sales	Sales Manager	2004	42000
8	Sales	Sales Manager	2005	450000
9	Sales	Sales Manager	2006	474000
10	Sales	Sales Manager	2007	507000
11	Sales	Sales Manager	2008	732000
12	Sales	Sales Manager	All years	2205000
13	Sales	Sales Representative	2004	348000
14	Sales	Sales Representative	2005	1045500
15	Sales	Sales Representative	2006	1849900
16	Sales	Sales Representative	2007	2400200
17	Sales	Sales Representative	2008	2949500
18	Sales	Sales Representative	All years	8593100

	DEPARTMENT_NAME	JOB	YEAR	TOTAL_SALARY
106	Administration	Administration As...	2007	52800
107	Administration	Administration As...	2008	52800
108	Administration	Administration As...	All years	264000
109	Administration	All Jobs	All years	264000
110	Human Resources	Human Resources R...	2004	78000
111	Human Resources	Human Resources R...	2005	78000
112	Human Resources	Human Resources R...	2006	78000
113	Human Resources	Human Resources R...	2007	78000
114	Human Resources	Human Resources R...	2008	78000
115	Human Resources	Human Resources R...	All years	390000
116	Human Resources	All Jobs	All years	390000
117	Public Relations	Public Relations ...	2004	120000
118	Public Relations	Public Relations ...	2005	120000
119	Public Relations	Public Relations ...	2006	120000
120	Public Relations	Public Relations ...	2007	120000
121	Public Relations	Public Relations ...	2008	120000
122	Public Relations	Public Relations ...	All years	600000
123	Public Relations	All Jobs	All years	600000

b. Reports with Rank and Percent_Rank:

REPORT 3

(a) Query.

Show top 5 total sales by each order mode and each month.

(b) Explanation on why such a query is necessary or useful for the management.

Because this query can help the management find the top five popular items sold by a specific mode in a specific time. For example, if the direct mode in September generate the highest total sales, the management can provide a promotion in this time through the direct mode to effectively increase the sales.

(c) SQL commands.

```
select *
from
(select o.order_mode_desc as order_mode, y.year_month_id as year_month, sum(sf.total_sales) as total_sales,
   rank() over (order by sum(sf.total_sales) desc) as rank
from order_mode_dim o, year_month_dim y, sales_order_fact sf
```

```
where sf.order_mode_id = o.order_mode_id
and sf.year_month_id = y.year_month_id
group by o.order_mode_desc, y.year_month_id) where rank <= 5;
```

(d) The screenshots of the query results.

	ORDER_MODE	YEAR_MONTH	TOTAL_SALES	RANK
1	direct	200709	367945.9	1
2	direct	200707	280645.1	2
3	online	200706	275958.08	3
4	online	200712	224305.2	4
5	online	200711	223456.4	5

REPORT 4

(a) Query.

Show top 30% number of employees by each department and by each country in May 2006.

(b) Explanation on why such a query is necessary or useful for the management.

It is useful for the management balance the number of recruiters. According to this query, the management can reduce hiring for those departments which have too many employees, while appropriately increase hiring for the rest departments.

(c) SQL commands.

```
select * from
(select d.department_name, c.country_name, sum(ef.number_of_employees) as number_of_employees,
   percent_rank() over (order by sum(ef.number_of_employees) desc) as percent_rank
from department_dim d, city_country_region_dim c, employee_fact ef
where d.department_id = ef.department_id and c.city_country_region_id = ef.city_country_region_id and
ef.year_month_id = 200605
group by d.department_name, c.country_name)
where percent_rank <= 0.3;
```

(d) The screenshots of the query results.

	DEPARTMENT_NAME	COUNTRY_NAME	NUMBER_OF_EMPLOYEES	PERCENT_RANK
1	Shipping	United States of America	25	0
2	Sales	United Kingdom	20	0.1
3	Finance	United States of America	5	0.2
4	Purchasing	United States of America	4	0.3

c. Reports with Partitions:

REPORT 5

(a) Query.

Show highest total salary in each city partition by month.

(b) Explanation on why such a query is necessary or useful for the management.

Because this query can show which city has the highest total salary in each month. Mangaer can use this information to effectively allocate the financial resources to each city which allocates a department. For example, if the highest salary was still in Seattle in 2004, but in the begin of 2005, the highest salary was in Oxford, so the management may appropriately adjust the following financial resources arrangement.

(c) SQL commands.

```
select *
from
(select ef.year_month_id, c.city, sum(total_salary) as total_salary,
   dense_rank() over (partition by year_month_id order by sum(total_salary) desc) as dense_rank
from employee_fact ef, city_country_region_dim c
where ef.city_country_region_id = c.city_country_region_id
group by c.city, ef.year_month_id)
where dense_rank = 1;
```

(d) The screenshots of the query results.

	YEAR_MONTH_ID	CITY	TOTAL_SALARY	DENSE_RANK
1	200401	Seattle	101816	1
2	200402	Seattle	101816	1
3	200403	Seattle	101816	1
4	200404	Seattle	101816	1
5	200405	Seattle	101816	1
6	200406	Seattle	101816	1
7	200407	Seattle	101816	1
8	200408	Seattle	101816	1
9	200409	Seattle	101816	1
10	200410	Seattle	101816	1
11	200411	Seattle	101816	1
12	200412	Seattle	101816	1
13	200501	Seattle	101816	1
14	200502	Seattle	101816	1
15	200503	Oxford	126800	1

	YEAR_MONTH_ID	CITY	TOTAL_SALARY	DENSE_RANK
45	200709	Oxford	244100	1
46	200710	Oxford	255100	1
47	200711	Oxford	262100	1
48	200712	Oxford	262100	1
49	200801	Oxford	286000	1
50	200802	Oxford	292800	1
51	200803	Oxford	299200	1
52	200804	Oxford	311500	1
53	200805	Oxford	311500	1
54	200806	Oxford	311500	1
55	200807	Oxford	311500	1
56	200808	Oxford	311500	1
57	200809	Oxford	311500	1
58	200810	Oxford	311500	1
59	200811	Oxford	311500	1
60	200812	Oxford	311500	1

REPORT 6

(a) Query.

Show ascending dense rank of total sales by each city partition by season.

(b) Explanation on why such a query is necessary or useful for the management.

This query is useful for the management adjust the product distribution structure in each season. Because season will make difference on customer purchasing option in difference citys. So manager could change the promotion strategy of each city in different season by this report. For example, if in the spring, the total sales in Detroit are much higher than in Elkhart, so the management can reduce supply in Elkhart or change the marketing ways to boost sales.

(c) SQL commands.

```
select s.season_desc as season, c.city, sum(sf.total_sales) as total_sales,
    dense_rank() over (partition by s.season_desc order by sum(sf.total_sales)) as dense_rank
from sales_order_fact sf, season_dim s, city_country_region_dim c
where sf.season_id = s.season_id and c.city_country_region_id = sf.city_country_region_id
group by s.season_desc, c.city;
```

(d) The screenshots of the query results.

	SEASON	CITY	TOTAL_SALES	DENSE_RANK
1	Autumn	Bloomington	2273.6	1
2	Autumn	Dubuque	25691.3	2
3	Autumn	Des Moines	27249.6	3
4	Autumn	Elkhart	31630	4
5	Autumn	Eau Claire	39684.7	5
6	Autumn	Indianapolis	40727.4	6
7	Autumn	Kokomo	48154.6	7
8	Autumn	Milwaukee	49799.3	8
9	Autumn	Madison	52758.9	9
10	Autumn	Cedar Rapids	71784.9	10
11	Autumn	Grand Rapids	79498.6	11
12	Autumn	South Bend	100081.2	12
13	Autumn	Minneapolis	128322.8	13
14	Spring	Elkhart	990.4	1
15	Spring	Detroit	19441.44	2

	SEASON	CITY	TOTAL_SALES	DENSE_RANK
36	Winter	La Crosse	1233	4
37	Winter	Cedar Rapids	5570.8	5
38	Winter	Detroit	8891.52	6
39	Winter	Ann Arbor	9106.48	7
40	Winter	Eau Claire	10640.8	8
41	Winter	Bloomington	22132.1	9
42	Winter	Beloit	26341	10
43	Winter	Davenport	30844.64	11
44	Winter	Indianapolis	50123.1	12
45	Winter	South Bend	91018.52	13
46	Winter	Grand Rapids	103780.1	14
47	Winter	Kokomo	112316.5	15
48	Winter	Elkhart	125429.7	16
49	Winter	Madison	165103.84	17
50	Winter	Milwaukee	379002.84	18

d. Reports with moving and cumulative aggregates:

REPORT 7

(a) Query.

Calculate each month sales and total sales in the latest 3 months of each month from 2004 Jan to 2008 Dec of each product from the listed three products ("KB 101/EN", "LaserPro 600/6/BW", "Screws <B.28.S>").

(b) Explanation on why such a query is necessary or useful for the management.

Because this query can help manager know whether there is a significant increase in sales for a specific product recently and which period is the best time to sale the product. Manager can judge whether the product is popular or not and modify the marketing strategy according to the past sales growth.

(c) SQL commands.

```
select a.product_id, a.year_month_id, sum(nvl(sf.total_sales, 0)) as month_sales,
   sum(sum(nvl(sf.total_sales, 0))) over (partition by a.product_id order by a.product_id, a.year_month_id
rows 2 preceding) as moving_3_month_total_sales
from (select distinct sf2.product_id, y.year_month_id
   from sales_order_fact sf2, year_month_dim y, product_dim p
   where y.year_month_id <= '200812'
   and y.year_month_id >= '200401'
   and p.product_id = sf2.product_id
   and p.product_name in ('KB 101/EN', 'LaserPro 600/6/BW', 'Screws <B.28.S>')) a
left join
   sales_order_fact sf
on sf.product_id = a.product_id and sf.year_month_id = a.year_month_id
group by a.product_id, a.year_month_id;
```

(d) The screenshots of the query results.

	PRODUCT_ID	YEAR_MONTH_ID	MONTH_SALES	MOVING_3_MONTH_TOTAL_SALES
1	3106	200401	0	0
2	3106	200402	0	0
3	3106	200403	0	0
4	3106	200404	0	0
5	3106	200405	0	0
6	3106	200406	0	0
7	3106	200407	0	0
8	3106	200408	0	0
9	3106	200409	0	0
10	3106	200410	0	0
11	3106	200411	0	0
12	3106	200412	0	0
13	3106	200501	0	0
14	3106	200502	0	0
15	3106	200503	0	0

	PRODUCT_ID	YEAR_MONTH_ID	MONTH_SALES	MOVING_3_MONTH_TOTAL_SALES
166	3143	200710	192	2256
167	3143	200711	187.2	379.2
168	3143	200712	0	379.2
169	3143	200801	0	187.2
170	3143	200802	0	0
171	3143	200803	0	0
172	3143	200804	0	0
173	3143	200805	2080	2080
174	3143	200806	1344	3424
175	3143	200807	848	4272
176	3143	200808	0	2192
177	3143	200809	0	848
178	3143	200810	0	0
179	3143	200811	0	0
180	3143	200812	0	0

REPORT 8

(a) Query.

Calculate each year salary and cumulative total salary in the before years of each year from 2004 to 2008.

(b) Explanation on why such a query is necessary or useful for the management.

Because this query can help the management to record the total salary for one country in a particular time period. And the management can estimate and modify the whole manual salary for a country in the next year according to the past salary and salary growth.

(c) SQL commands.

```
select c.country_name, ym.year, sum(ef.total_salary) as year_salary,
    sum(sum(ef.total_salary)) over (partition by c.country_name order by c.country_name, ym.year rows unbounded preceding) as cumulative_total_salary
from employee_fact ef, year_month_dim ym, city_country_region_dim c
where ef.year_month_id = ym.year_month_id and ef.city_country_region_id = c.city_country_region_id
group by c.country_name, ym.year
order by c.country_name, ym.year;
```

(d) The screenshots of the query results.

	COUNTRY_NAME	YEAR	YEAR_SALARY	CUMULATIVE_TOTAL_SALARY
1	Canada	2004	143000	143000
2	Canada	2005	186000	329000
3	Canada	2006	228000	557000
4	Canada	2007	228000	785000
5	Canada	2008	228000	1013000
6	Germany	2004	120000	120000
7	Germany	2005	120000	240000
8	Germany	2006	120000	360000
9	Germany	2007	120000	480000
10	Germany	2008	120000	600000
11	United Kingdom	2004	468000	468000
12	United Kingdom	2005	1573500	2041500
13	United Kingdom	2006	2401900	4443400
14	United Kingdom	2007	2985200	7428600
15	United Kingdom	2008	3759500	11188100
16	United States of America	2004	1567292	1567292
17	United States of America	2005	2161092	3728384
18	United States of America	2006	3243892	6972276
19	United States of America	2007	3715092	10687368
20	United States of America	2008	4097192	14784560

Task 5 Execution Plan

Query 1

Calculate the total sales of all order modes and all seasons, the subtotal sales of all order modes in each season, the subtotal sales of all seasons by each order mode and the subtotal sales of the combination of each season and each order mode.

(a) SQL from task 4

```
SELECT
    DECODE(
        GROUPING(o.order_mode_desc),
        1,
        'All Order Modes',
        o.order_mode_desc
    ) AS order_mode,
    DECODE(
        GROUPING(s.season_desc),
        1,
        'All Seasons',
        s.season_desc
    ) AS season,
    nvl(
        SUM(sf.total_sales),
        0
    ) AS total_sales
FROM
    sales_order_fact sf,
    order_mode_dim o,
    season_dim s
WHERE
        sf.season_id (+) = s.season_id
    AND
        sf.order_mode_id (+) = o.order_mode_id
GROUP BY
    CUBE(o.order_mode_desc,s.season_desc);
```

(b) Task4 query result

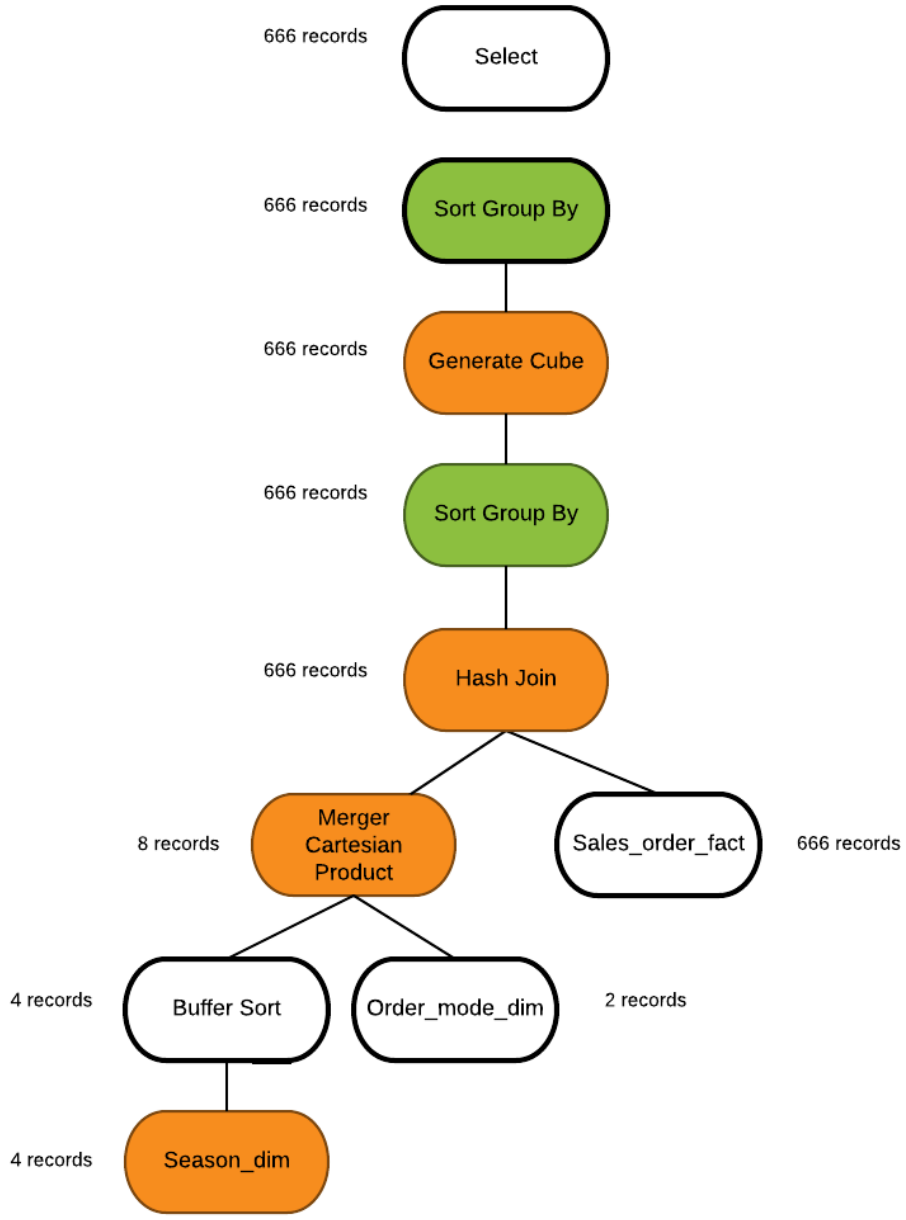
	ORDER_MODE	SEASON	TOTAL_SALES
1	All Order Modes	All Seasons	3478727.31
2	All Order Modes	Autumn	697656.9
3	All Order Modes	Spring	872398.08
4	All Order Modes	Summer	765232.19
5	All Order Modes	Winter	1143440.14
6	direct	All Seasons	1910487.05
7	direct	Autumn	371278.8
8	direct	Spring	638627.48
9	direct	Summer	285255.79
10	direct	Winter	615324.98
11	online	All Seasons	1568240.26
12	online	Autumn	326378.1
13	online	Spring	233770.6
14	online	Summer	479976.4
15	online	Winter	528115.16

(c) Task 4 query execution plan

PLAN_TABLE_OUTPUT

```
Plan hash value: 132443277

-------------------------------------------------------------------------------------------------
| Id  | Operation                | Name             | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |                  |   666 | 29970 |    12   (9)| 00:00:01 |
|   1 |  SORT GROUP BY           |                  |   666 | 29970 |    12   (9)| 00:00:01 |
|   2 |   GENERATE CUBE          |                  |   666 | 29970 |    12   (9)| 00:00:01 |
|   3 |    SORT GROUP BY         |                  |   666 | 29970 |    12   (9)| 00:00:01 |
|*  4 |     HASH JOIN OUTER      |                  |   666 | 29970 |    11   (0)| 00:00:01 |
|   5 |      MERGE JOIN CARTESIAN|                  |     8 |   280 |     7   (0)| 00:00:01 |
|   6 |       TABLE ACCESS FULL  | ORDER_MODE_DIM   |     2 |    30 |     3   (0)| 00:00:01 |
|   7 |       BUFFER SORT        |                  |     4 |    80 |     4   (0)| 00:00:01 |
|   8 |        TABLE ACCESS FULL | SEASON_DIM       |     4 |    80 |     2   (0)| 00:00:01 |
|   9 |      TABLE ACCESS FULL   | SALES_ORDER_FACT |   666 |  6660 |     4   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   4 - access("SF"."SEASON_ID"(+)="S"."SEASON_ID" AND
              "SF"."ORDER_MODE_ID"(+)="O"."ORDER_MODE_ID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

(d) Task4 query tree



(e) New SQL

```
SELECT /*+ USE_NL (o s) */
    DECODE(
        GROUPING(o.order_mode_desc),
        1,
        'All Order Modes',
        o.order_mode_desc
    ) AS order_mode,
```

```
    DECODE(
        GROUPING(s.season_desc),
        1,
        'All Seasons',
        s.season_desc
    ) AS season,
    nvl(
        SUM(sf.total_sales),
        0
    ) AS total_sales
FROM
    sales_order_fact sf,
    order_mode_dim o,
    season_dim s
WHERE
        sf.season_id (+) = s.season_id
    AND
        sf.order_mode_id (+) = o.order_mode_id
GROUP BY
    CUBE(o.order_mode_desc,s.season_desc);
```

(f) New query result

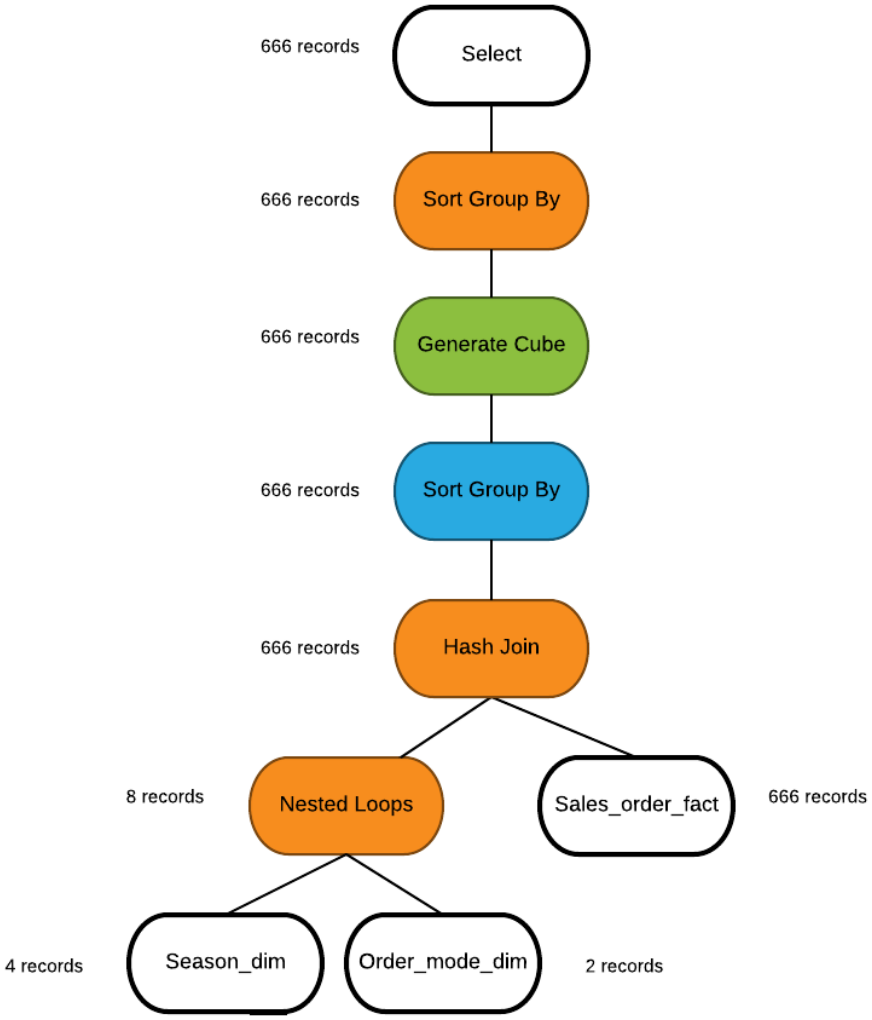
	ORDER_MODE	SEASON	TOTAL_SALES
1	All Order Modes	All Seasons	3478727.31
2	All Order Modes	Autumn	697656.9
3	All Order Modes	Spring	872398.08
4	All Order Modes	Summer	765232.19
5	All Order Modes	Winter	1143440.14
6	direct	All Seasons	1910487.05
7	direct	Autumn	371278.8
8	direct	Spring	638627.48
9	direct	Summer	285255.79
10	direct	Winter	615324.98
11	online	All Seasons	1568240.26
12	online	Autumn	326378.1
13	online	Spring	233770.6
14	online	Summer	479976.4
15	online	Winter	528115.16

(g) New query execution plan

PLAN_TABLE_OUTPUT

```
Plan hash value: 52099599

-----------------------------------------------------------------------------------------
| Id  | Operation              | Name             | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |                  |   666 | 29970 |    12   (9)| 00:00:01 |
|   1 |  SORT GROUP BY         |                  |   666 | 29970 |    12   (9)| 00:00:01 |
|   2 |   GENERATE CUBE        |                  |   666 | 29970 |    12   (9)| 00:00:01 |
|   3 |    SORT GROUP BY       |                  |   666 | 29970 |    12   (9)| 00:00:01 |
|*  4 |     HASH JOIN OUTER    |                  |   666 | 29970 |    11   (0)| 00:00:01 |
|   5 |      NESTED LOOPS      |                  |     8 |   280 |     7   (0)| 00:00:01 |
|   6 |       TABLE ACCESS FULL| ORDER_MODE_DIM   |     2 |    30 |     3   (0)| 00:00:01 |
|   7 |       TABLE ACCESS FULL| SEASON_DIM       |     4 |    80 |     2   (0)| 00:00:01 |
|   8 |      TABLE ACCESS FULL | SALES_ORDER_FACT |   666 |  6660 |     4   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   4 - access("SF"."SEASON_ID"(+)="S"."SEASON_ID" AND
              "SF"."ORDER_MODE_ID"(+)="O"."ORDER_MODE_ID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

(h) New query tree



(i) Comparison

	Task4 query	New query
Execution time	9	8
Which is better		√
Join Type	Cartesian Product	Nest loop
Reason	These two ways are all used to do a Cartesian Product operation, but in task4 query, two dimension tables need to be processed first to use cartesian product, and table must be sorted first using “Buffer Sort” operation to sort records, this step use more time then nest loop.	

Query 2

Calculate the total salary of each department, the subtotal salary of each department by each job in all years, the subtotal salary of each departments with a combination of each job and each year

(a) SQL from task 4

```
SELECT
    d.department_name,
    DECODE(
        GROUPING(j.job_title),
        1,
        'All Jobs',
        j.job_title
    ) AS job,
    DECODE(
        GROUPING(y.year),
        1,
        'All years',
        y.year
    ) AS year,
    SUM(ef.total_salary) AS total_salary
FROM
    employee_fact ef,
    department_dim d,
    job_dim j,
    year_month_dim y
WHERE
        j.job_id = ef.job_id
    AND
        ef.department_id = d.department_id
    AND
        y.year_month_id = ef.year_month_id
GROUP BY
    d.department_name,
    ROLLUP(j.job_title,y.year);
```

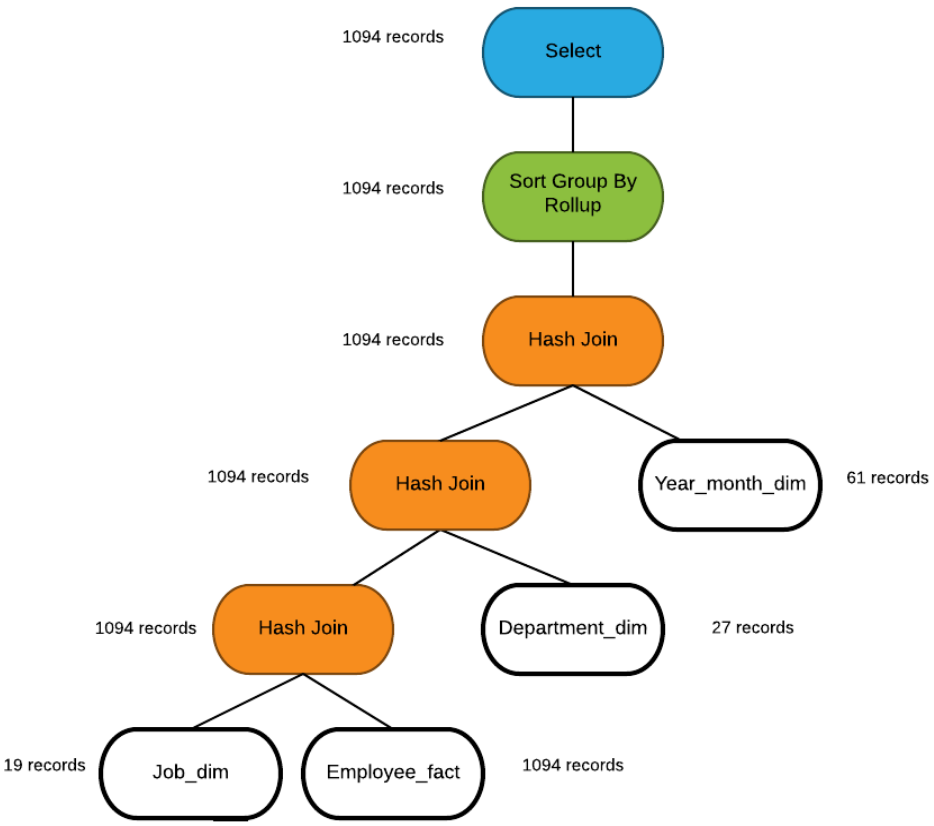
(b)Task4 query result

	DEPARTMENT_NAME	JOB	YEAR	TOTAL_SALARY
1	IT	Programmer	2005	33600
2	IT	Programmer	2006	218400
3	IT	Programmer	2007	317400
4	IT	Programmer	2008	345600
5	IT	Programmer	All years	915000
6	IT	All Jobs	All years	915000
7	Sales	Sales Manager	2004	42000
8	Sales	Sales Manager	2005	450000
9	Sales	Sales Manager	2006	474000
10	Sales	Sales Manager	2007	507000
11	Sales	Sales Manager	2008	732000
12	Sales	Sales Manager	All years	2205000
13	Sales	Sales Representative	2004	348000
14	Sales	Sales Representative	2005	1045500
15	Sales	Sales Representative	2006	1849900
16	Sales	Sales Representative	2007	2400200
17	Sales	Sales Representative	2008	2949500
18	Sales	Sales Representative	All years	8593100
19	Sales	All Jobs	All years	10798100
20	Finance	Accountant	2004	108000
21	Finance	Accountant	2005	171600
22	Finance	Accountant	2006	376800
23	Finance	Accountant	2007	399300
24	Finance	Accountant	2008	475200
25	Finance	Accountant	All years	1530900
26	Finance	Finance Manager	2004	144096
27	Finance	Finance Manager	2005	144096
28	Finance	Finance Manager	2006	144096
29	Finance	Finance Manager	2007	144096
30	Finance	Finance Manager	2008	144096

(c) Task 4 query execution plan

```
PLAN_TABLE_OUTPUT
Plan hash value: 4220686471

-------------------------------------------------------------------------------------------
| Id  | Operation             | Name            | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |                 |  1094 | 85332 |    14   (8)| 00:00:01 |
|   1 |  SORT GROUP BY ROLLUP |                 |  1094 | 85332 |    14   (8)| 00:00:01 |
|*  2 |   HASH JOIN           |                 |  1094 | 85332 |    13   (0)| 00:00:01 |
|   3 |    TABLE ACCESS FULL  | YEAR_MONTH_DIM  |    61 |   732 |     3   (0)| 00:00:01 |
|*  4 |    HASH JOIN          |                 |  1094 | 72204 |    10   (0)| 00:00:01 |
|   5 |     TABLE ACCESS FULL | DEPARTMENT_DIM  |    27 |   432 |     3   (0)| 00:00:01 |
|*  6 |     HASH JOIN         |                 |  1094 | 54700 |     7   (0)| 00:00:01 |
|   7 |      TABLE ACCESS FULL| JOB_DIM         |    19 |   513 |     3   (0)| 00:00:01 |
|   8 |      TABLE ACCESS FULL| EMPLOYEE_FACT   |  1094 | 25162 |     4   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("Y"."YEAR_MONTH_ID"="EF"."YEAR_MONTH_ID")
   4 - access("EF"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
   6 - access("J"."JOB_ID"="EF"."JOB_ID")
```

(d) Task4 query tree



(e) New SQL

```
SELECT /*+ ORDERED */
    d.department_name,
    DECODE(
        GROUPING(j.job_title),
        1,
        'All Jobs',
        j.job_title
    ) AS job,
    DECODE(
        GROUPING(y.year),
        1,
        'All years',
        y.year
    ) AS year,
    SUM(ef.total_salary) AS total_salary
FROM
    employee_fact ef,
    department_dim d,
    job_dim j,
    year_month_dim y
WHERE
        j.job_id = ef.job_id
    AND
        ef.department_id = d.department_id
    AND
        y.year_month_id = ef.year_month_id
GROUP BY
    d.department_name,
    ROLLUP(j.job_title,y.year);
```

(f) New query result

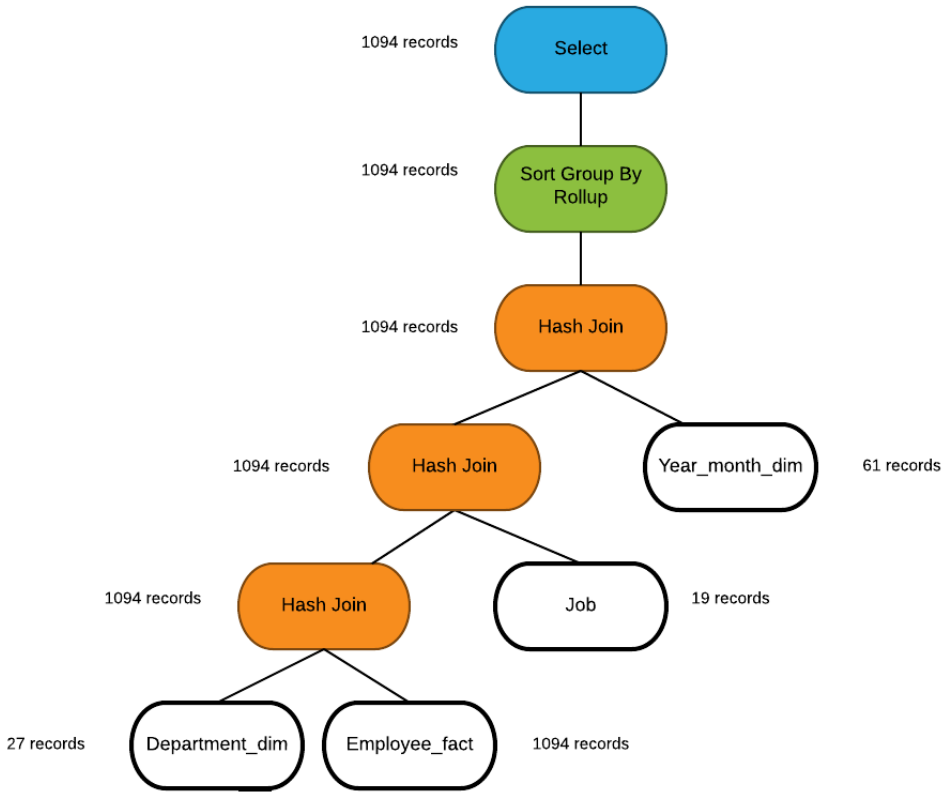
	DEPARTMENT_NAME	JOB	YEAR	TOTAL_SALARY
1	IT	Programmer	2005	33600
2	IT	Programmer	2006	218400
3	IT	Programmer	2007	317400
4	IT	Programmer	2008	345600
5	IT	Programmer	All years	915000
6	IT	All Jobs	All years	915000
7	Sales	Sales Manager	2004	42000
8	Sales	Sales Manager	2005	450000
9	Sales	Sales Manager	2006	474000
10	Sales	Sales Manager	2007	507000
11	Sales	Sales Manager	2008	732000
12	Sales	Sales Manager	All years	2205000
13	Sales	Sales Representative	2004	348000
14	Sales	Sales Representative	2005	1045500
15	Sales	Sales Representative	2006	1849900
16	Sales	Sales Representative	2007	2400200
17	Sales	Sales Representative	2008	2949500
18	Sales	Sales Representative	All years	8593100
19	Sales	All Jobs	All years	10798100
20	Finance	Accountant	2004	108000
21	Finance	Accountant	2005	171600
22	Finance	Accountant	2006	376800
23	Finance	Accountant	2007	399300
24	Finance	Accountant	2008	475200
25	Finance	Accountant	All years	1530900
26	Finance	Finance Manager	2004	144096
27	Finance	Finance Manager	2005	144096
28	Finance	Finance Manager	2006	144096
29	Finance	Finance Manager	2007	144096
30	Finance	Finance Manager	2008	144096
31	Finance	Finance Manager	All years	720480
32	Finance	All Jobs	All years	2251380
33	Shipping	Stock Clerk	2004	108300
34	Shipping	Stock Clerk	2005	243200

(g) New query execution plan

PLAN_TABLE_OUTPUT

```
Plan hash value: 1000859412

-------------------------------------------------------------------------------------------
| Id  | Operation             | Name            | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |                 |  1094 | 85332 |    14   (8)| 00:00:01 |
|   1 |  SORT GROUP BY ROLLUP |                 |  1094 | 85332 |    14   (8)| 00:00:01 |
|*  2 |   HASH JOIN           |                 |  1094 | 85332 |    13   (0)| 00:00:01 |
|   3 |    TABLE ACCESS FULL  | YEAR_MONTH_DIM  |    61 |   732 |     3   (0)| 00:00:01 |
|*  4 |    HASH JOIN          |                 |  1094 | 72204 |    10   (0)| 00:00:01 |
|   5 |     TABLE ACCESS FULL | JOB_DIM         |    19 |   513 |     3   (0)| 00:00:01 |
|*  6 |     HASH JOIN         |                 |  1094 | 42666 |     7   (0)| 00:00:01 |
|   7 |      TABLE ACCESS FULL| EMPLOYEE_FACT   |  1094 | 25162 |     4   (0)| 00:00:01 |
|   8 |      TABLE ACCESS FULL| DEPARTMENT_DIM  |    27 |   432 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("Y"."YEAR_MONTH_ID"="EF"."YEAR_MONTH_ID")
   4 - access("J"."JOB_ID"="EF"."JOB_ID")
   6 - access("EF"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
```

(h) New query tree



(i) Comparison

	Task4 query	New query
Execution time	8	8
Which is better	√	√
Join Type	Hash Join	Hash Join
Reason	In these two queries, hash join is used to join four tables, which is the best way to join the small size tables, so I just use ordered hint to change the order of hash join, but the result shows that the outcome is the same.	

Query 3

Show top 5 total sales by each order mode and each month.

(a) SQL from task 4

```
SELECT
    *
FROM
    (
        SELECT
            o.order_mode_desc AS order_mode,
            y.year_month_id AS year_month,
            SUM(sf.total_sales) AS total_sales,
            RANK() OVER(
                ORDER BY SUM(sf.total_sales) DESC
            ) AS rank
        FROM
            order_mode_dim o,
            year_month_dim y,
            sales_order_fact sf
        WHERE
                sf.order_mode_id = o.order_mode_id
            AND
                sf.year_month_id = y.year_month_id
        GROUP BY
            o.order_mode_desc,
            y.year_month_id
    )
WHERE
    rank <= 5;
```

(b) Task4 query result

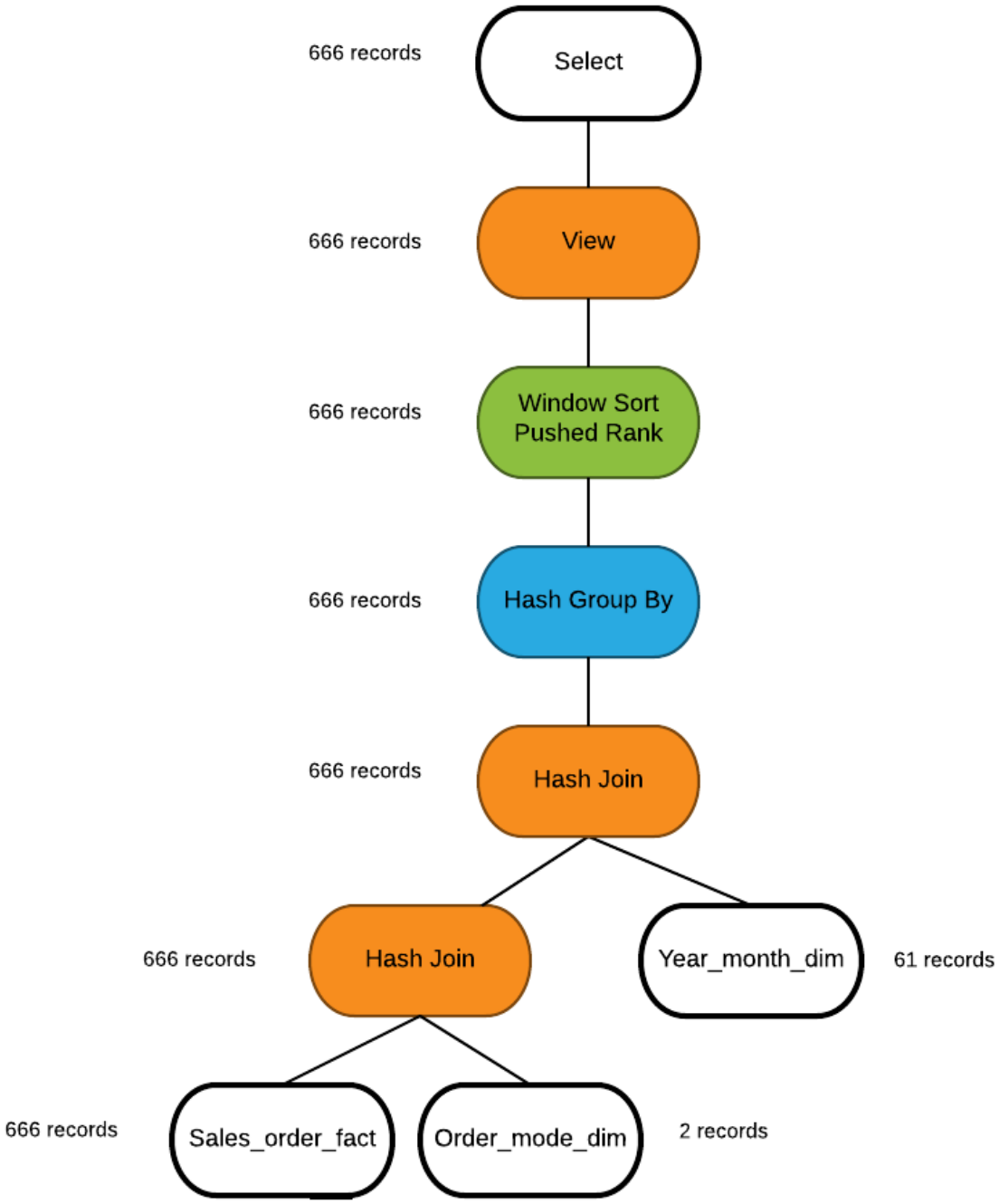
	ORDER_MODE	YEAR_MONTH	TOTAL_SALES	RANK
1	direct	200709	367945.9	1
2	direct	200707	280645.1	2
3	online	200706	275958.08	3
4	online	200712	224305.2	4
5	online	200711	223456.4	5

(c) Task 4 query execution plan

PLAN_TABLE_OUTPUT

```
Plan hash value: 2052129035

-------------------------------------------------------------------------------------------
| Id  | Operation                | Name              | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |                   |    25 |  1075 |    12  (17)| 00:00:01 |
|*  1 |  VIEW                    |                   |    25 |  1075 |    12  (17)| 00:00:01 |
|*  2 |   WINDOW SORT PUSHED RANK|                   |    25 |   900 |    12  (17)| 00:00:01 |
|   3 |    HASH GROUP BY         |                   |    25 |   900 |    12  (17)| 00:00:01 |
|*  4 |     HASH JOIN            |                   |   666 | 23976 |    10   (0)| 00:00:01 |
|   5 |      TABLE ACCESS FULL   | YEAR_MONTH_DIM    |    61 |   427 |     3   (0)| 00:00:01 |
|*  6 |      HASH JOIN           |                   |   666 | 19314 |     7   (0)| 00:00:01 |
|   7 |       TABLE ACCESS FULL  | ORDER_MODE_DIM    |     2 |    30 |     3   (0)| 00:00:01 |
|   8 |       TABLE ACCESS FULL  | SALES_ORDER_FACT  |   666 |  9324 |     4   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("RANK"<=5)
   2 - filter(RANK() OVER ( ORDER BY SUM("SF"."TOTAL_SALES") DESC )<=5)
   4 - access("SF"."YEAR_MONTH_ID"="Y"."YEAR_MONTH_ID")
   6 - access("SF"."ORDER_MODE_ID"="O"."ORDER_MODE_ID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

(d) Task4 query tree



(e) New SQL

```
SELECT
    *
FROM
    (
        SELECT    /*+ ORDERED USE_NL (o y) */
            o.order_mode_desc AS order_mode,
            y.year_month_id AS year_month,
            SUM(sf.total_sales) AS total_sales,
            RANK() OVER(
                ORDER BY SUM(sf.total_sales) DESC
            ) AS rank
        FROM
            order_mode_dim o,
            sales_order_fact sf,
            year_month_dim y
        WHERE
                sf.order_mode_id = o.order_mode_id
            AND
                sf.year_month_id = y.year_month_id
        GROUP BY
            o.order_mode_desc,
            y.year_month_id
    )
WHERE
    rank <= 5;
```

(f) New query result

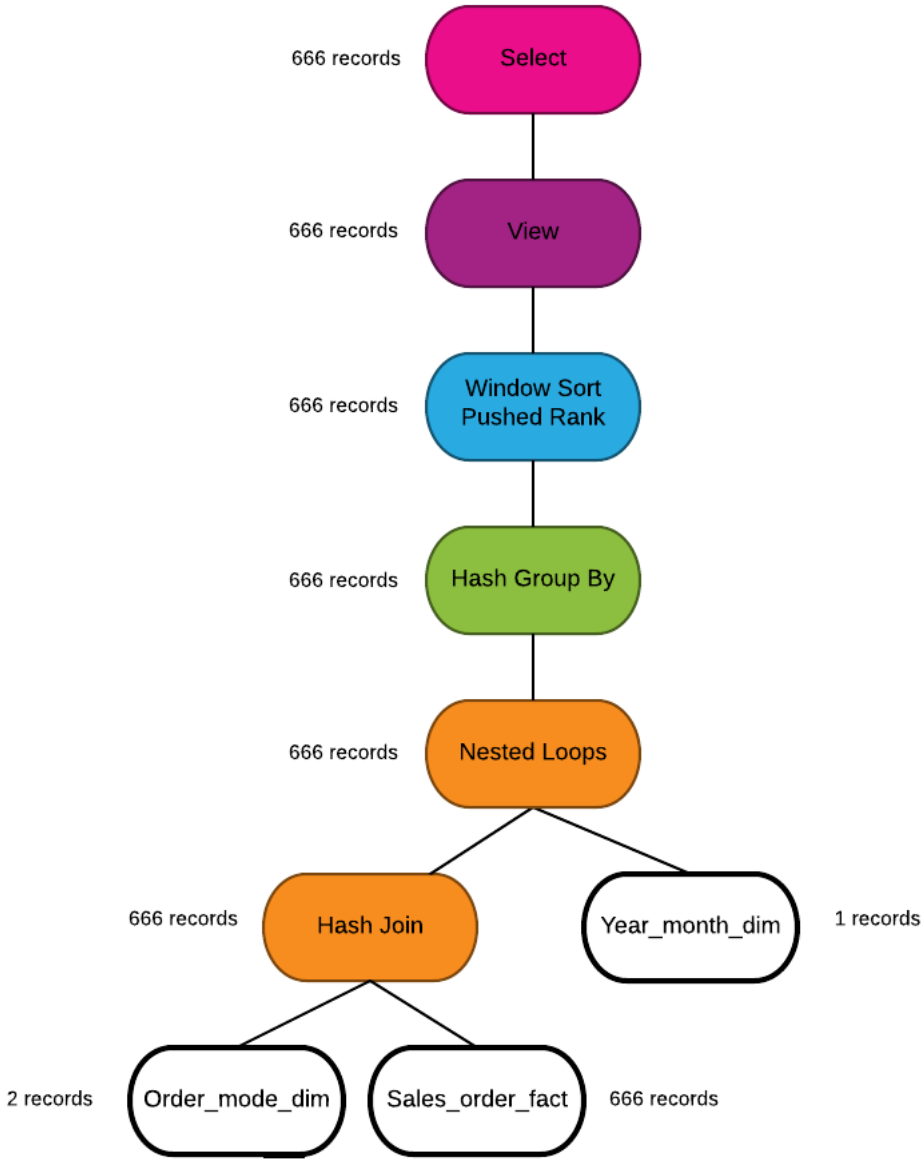
	ORDER_MODE	YEAR_MONTH	TOTAL_SALES	RANK
1	direct	200709	367945.9	1
2	direct	200707	280645.1	2
3	online	200706	275958.08	3
4	online	200712	224305.2	4
5	online	200711	223456.4	5

(g) New query execution plan

PLAN_TABLE_OUTPUT

```
Plan hash value: 3507033831

-------------------------------------------------------------------------------------...
| Id  | Operation                | Name              | Rows  | Bytes | Cost (%CPU)| Tim...
-------------------------------------------------------------------------------------...
|   0 | SELECT STATEMENT         |                   |    25 |  1075 |   733   (1)| 00:...
|*  1 |  VIEW                    |                   |    25 |  1075 |   733   (1)| 00:...
|*  2 |   WINDOW SORT PUSHED RANK|                   |    25 |   900 |   733   (1)| 00:...
|   3 |    HASH GROUP BY         |                   |    25 |   900 |   733   (1)| 00:...
|   4 |     NESTED LOOPS         |                   |   666 | 23976 |   731   (1)| 00:...
|*  5 |      HASH JOIN           |                   |   666 | 19314 |     7   (0)| 00:...
|   6 |       TABLE ACCESS FULL  | ORDER_MODE_DIM    |     2 |    30 |     3   (0)| 00:...
|   7 |       TABLE ACCESS FULL  | SALES_ORDER_FACT  |   666 |  9324 |     4   (0)| 00:...
|*  8 |      TABLE ACCESS FULL   | YEAR_MONTH_DIM    |     1 |     7 |     1   (0)| 00:...
-------------------------------------------------------------------------------------...

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("RANK"<=5)
   2 - filter(RANK() OVER ( ORDER BY SUM("SF"."TOTAL_SALES") DESC )<=5)
   5 - access("SF"."ORDER_MODE_ID"="O"."ORDER_MODE_ID")
   8 - filter("SF"."YEAR_MONTH_ID"="Y"."YEAR_MONTH_ID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

(h) New query tree



(i) Comparison

	Task4 query	New query
Execution time	8	8
Which is better	√	
Join Type	Hash join	Nest loop, ordered
Reason	The task4 query use hash join to join three tables which should be the best way, the result of nest loop shows fewer records, but because nest loop need to use loop to scan record in one table to each of record in another table to match the attribute, so it occupy large proportion of CPU, is not efficient.	

Query 4

Show top 30% number of employees by each department and by each country in May 2006.

(a) SQL from task 4

```
SELECT
    *
FROM
    (
        SELECT
            d.department_name,
            c.country_name,
            SUM(ef.number_of_employees) AS number_of_employees,
            PERCENT_RANK() OVER(
                ORDER BY SUM(ef.number_of_employees) DESC
            ) AS percent_rank
        FROM
            department_dim d,
            city_country_region_dim c,
            employee_fact ef
        WHERE
                d.department_id = ef.department_id
            AND
                c.city_country_region_id = ef.city_country_region_id
            AND
                ef.year_month_id = 200605
        GROUP BY
            d.department_name,
            c.country_name
    )
WHERE
    percent_rank <= 0.3;
```

(b) Task4 query result

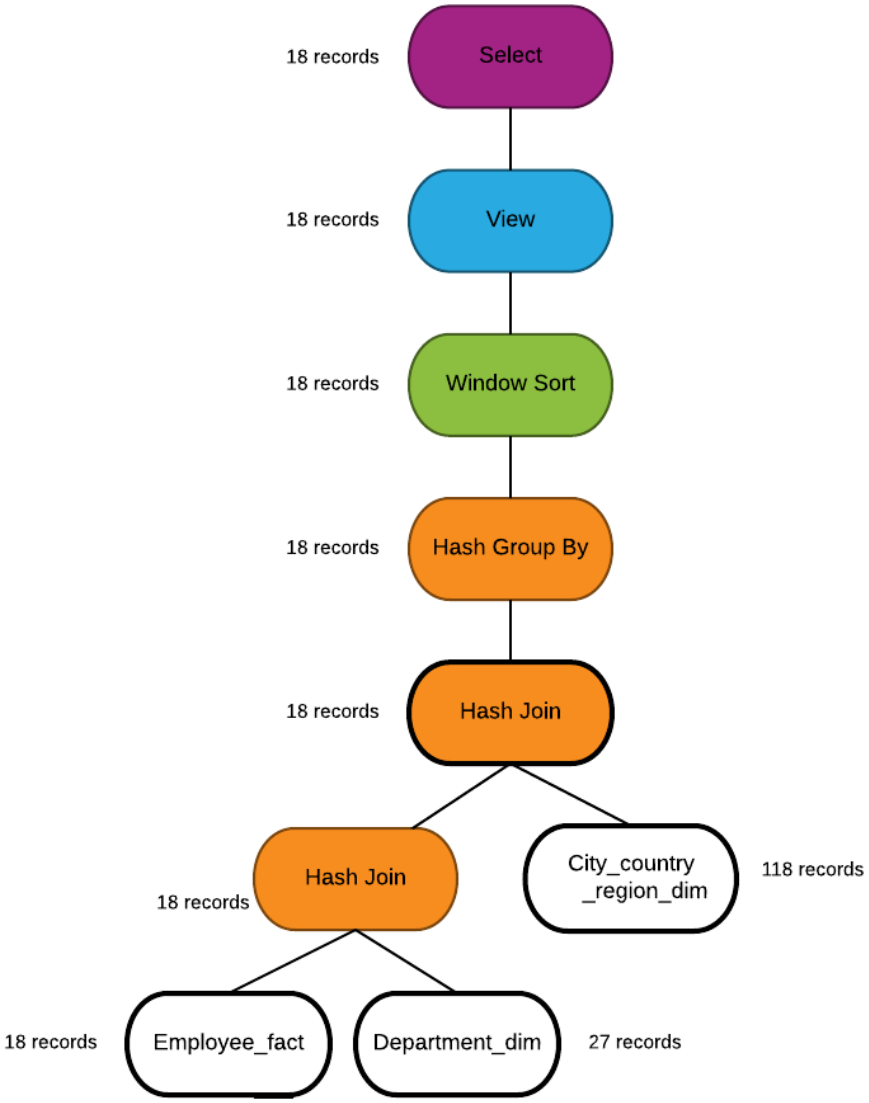
	DEPARTMENT_NAME	COUNTRY_NAME	NUMBER_OF_EMPLOYEES	PERCENT_RANK
1	Shipping	United States of America	25	0
2	Sales	United Kingdom	20	0.1
3	Finance	United States of America	5	0.2
4	Purchasing	United States of America	4	0.3

(c) Task 4 query execution plan

PLAN_TABLE_OUTPUT

```
Plan hash value: 819018354

-----------------------------------------------------------------------------------------------------------
| Id  | Operation              | Name                    | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |                         |    18 |  1170 |    12  (17)| 00:00:01 |
|*  1 |  VIEW                  |                         |    18 |  1170 |    12  (17)| 00:00:01 |
|   2 |   WINDOW SORT          |                         |    18 |  1404 |    12  (17)| 00:00:01 |
|   3 |    HASH GROUP BY       |                         |    18 |  1404 |    12  (17)| 00:00:01 |
|*  4 |     HASH JOIN          |                         |    18 |  1404 |    10   (0)| 00:00:01 |
|*  5 |      HASH JOIN         |                         |    18 |   810 |     7   (0)| 00:00:01 |
|*  6 |       TABLE ACCESS FULL| EMPLOYEE_FACT           |    18 |   522 |     4   (0)| 00:00:01 |
|   7 |       TABLE ACCESS FULL| DEPARTMENT_DIM          |    27 |   432 |     3   (0)| 00:00:01 |
|   8 |      TABLE ACCESS FULL | CITY_COUNTRY_REGION_DIM |   118 |  3894 |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("PERCENT_RANK"<=0.3)
   4 - access("C"."CITY_COUNTRY_REGION_ID"="EF"."CITY_COUNTRY_REGION_ID")
   5 - access("D"."DEPARTMENT_ID"="EF"."DEPARTMENT_ID")
   6 - filter(TO_NUMBER("EF"."YEAR_MONTH_ID")=200605)
```

(d) Task4 query tree



(e) New SQL

```
CREATE INDEX department_id_idx1 ON
    department_dim ( department_id );

CREATE INDEX department_id_idx2 ON
    employee_fact ( department_id );

SELECT
    *
FROM
    (
```

```
        SELECT /*+no_merge */
            d.department_name,
            c.country_name,
            SUM(ef.number_of_employees) AS number_of_employees,
            PERCENT_RANK() OVER(
                ORDER BY SUM(ef.number_of_employees) DESC
            ) AS percent_rank
        FROM
            department_dim d,
            city_country_region_dim c,
            employee_fact ef
        WHERE
                d.department_id = ef.department_id
            AND
                c.city_country_region_id = ef.city_country_region_id
            AND
                ef.year_month_id = 200605
        GROUP BY
            d.department_name,
            c.country_name
    )
WHERE
    percent_rank <= 0.3;
```

(f) New query result

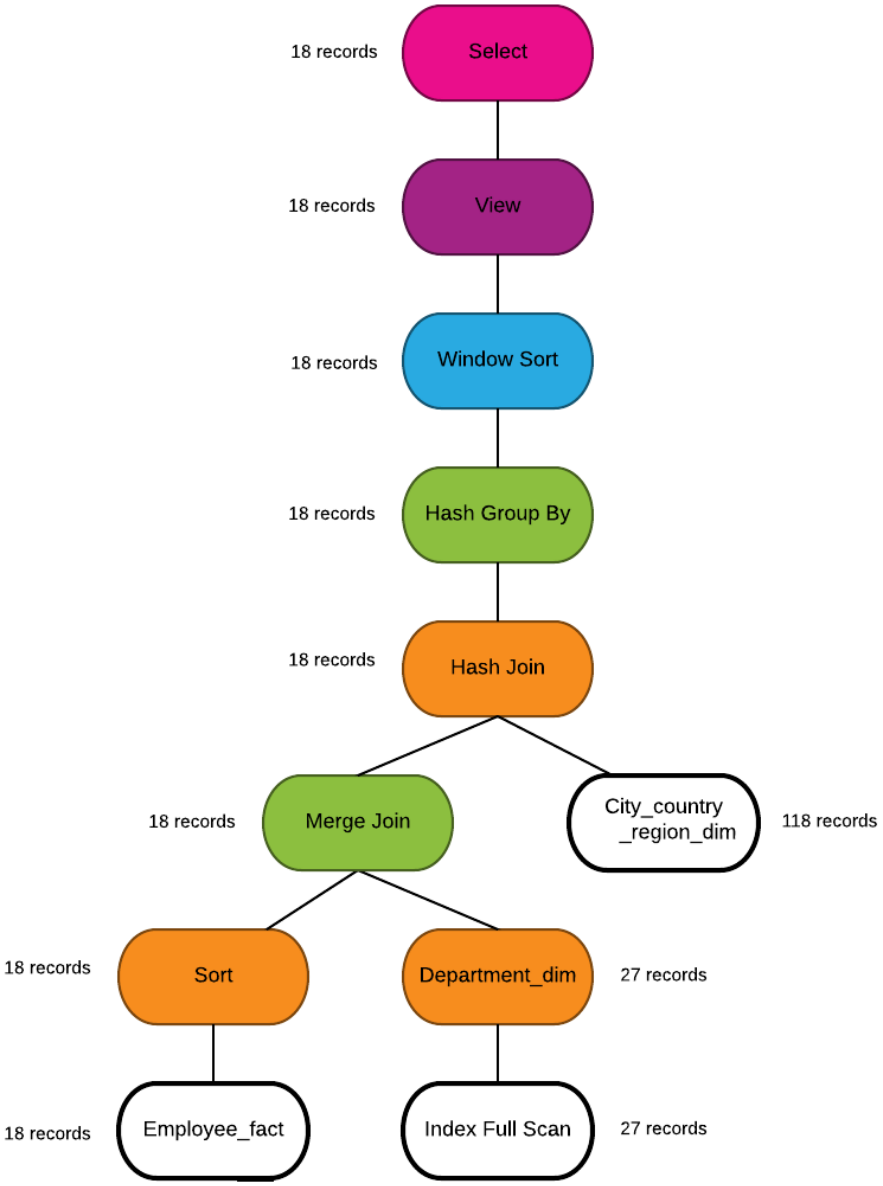
	DEPARTMENT_NAME	COUNTRY_NAME	NUMBER_OF_EMPLOYEES	PERCENT_RANK
1	Shipping	United States of America	25	0
2	Sales	United Kingdom	20	0.1
3	Finance	United States of America	5	0.2
4	Purchasing	United States of America	4	0.3

(g) New query execution plan

PLAN_TABLE_OUTPUT

```
Plan hash value: 2385089379

-------------------------------------------------------------------------------------...
| Id  | Operation                      | Name                    | Rows  | Bytes | C...
-------------------------------------------------------------------------------------...
|   0 | SELECT STATEMENT               |                         |    18 |  1170 |  ...
|*  1 |  VIEW                          |                         |    18 |  1170 |  ...
|   2 |   WINDOW SORT                  |                         |    18 |  1404 |  ...
|   3 |    HASH GROUP BY               |                         |    18 |  1404 |  ...
|*  4 |     HASH JOIN                  |                         |    18 |  1404 |  ...
|   5 |      MERGE JOIN                |                         |    18 |   810 |  ...
|   6 |       TABLE ACCESS BY INDEX ROWID| DEPARTMENT_DIM          |    27 |   432 |  ...
|   7 |        INDEX FULL SCAN         | DEPARTMENT_ID_IDX1      |    27 |       |  ...
|*  8 |       SORT JOIN                |                         |    18 |   522 |  ...
|*  9 |        TABLE ACCESS FULL       | EMPLOYEE_FACT           |    18 |   522 |  ...
|  10 |      TABLE ACCESS FULL         | CITY_COUNTRY_REGION_DIM |   118 |  3894 |  ...
-------------------------------------------------------------------------------------...

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("PERCENT_RANK"<=0.3)
   4 - access("C"."CITY_COUNTRY_REGION_ID"="EF"."CITY_COUNTRY_REGION_ID")
   8 - access("D"."DEPARTMENT_ID"="EF"."DEPARTMENT_ID")
       filter("D"."DEPARTMENT_ID"="EF"."DEPARTMENT_ID")
   9 - filter(TO_NUMBER("EF"."YEAR_MONTH_ID")=200605)
```

(h) New query tree



(i) Comparison

	Task4 query	New query
Execution time	8	10
Which is better	√	
Join Type	Hash join	Index scan, Merge join
Reason	In the new query, the department_id index is used, it will access index instead of scanning the full tale, which reduce the cost of CPU and improve the efficiency. However, it takes more steps than hash join.	

Query 5

Show highest total salary in each city partition by month.

(a) SQL from task 4

```
SELECT
    *
FROM
    (
        SELECT
            ef.year_month_id,
            c.city,
            SUM(total_salary) AS total_salary,
            DENSE_RANK() OVER(PARTITION BY
                year_month_id
                ORDER BY
                    SUM(total_salary)
                DESC
            ) AS dense_rank
        FROM
            employee_fact ef,
            city_country_region_dim c
        WHERE
            ef.city_country_region_id = c.city_country_region_id
        GROUP BY
            c.city,
            ef.year_month_id
    )
WHERE
    dense_rank = 1;
```

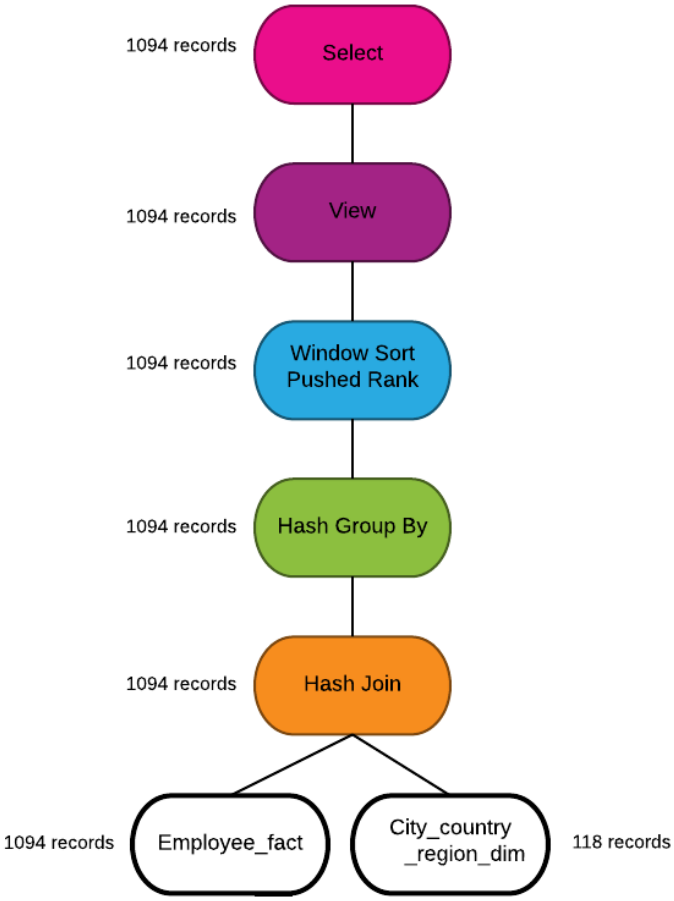
(b) Task4 query result

	YEAR_MONTH_ID	CITY	TOTAL_SALARY	DENSE_RANK
1	200401	Seattle	101816	1
2	200402	Seattle	101816	1
3	200403	Seattle	101816	1
4	200404	Seattle	101816	1
5	200405	Seattle	101816	1
6	200406	Seattle	101816	1
7	200407	Seattle	101816	1
8	200408	Seattle	101816	1
9	200409	Seattle	101816	1
10	200410	Seattle	101816	1
11	200411	Seattle	101816	1
12	200412	Seattle	101816	1
13	200501	Seattle	101816	1
14	200502	Seattle	101816	1
15	200503	Oxford	126800	1
16	200504	Oxford	126800	1
17	200505	Oxford	126800	1
18	200506	Oxford	126800	1
19	200507	Oxford	126800	1
20	200508	Oxford	135800	1
21	200509	Seattle	137516	1
22	200510	Seattle	137516	1
23	200511	Oxford	146300	1
24	200512	Oxford	153800	1
25	200601	Oxford	163400	1
26	200602	Oxford	163400	1
27	200603	Oxford	190000	1
28	200604	Oxford	198400	1
29	200605	Oxford	198400	1
30	200606	Oxford	198400	1
31	200607	Oxford	198400	1
32	200608	Oxford	198400	1
33	200609	Oxford	198400	1
34	200610	Oxford	198400	1

(c) Task 4 query execution plan

```
PLAN_TABLE_OUTPUT
Plan hash value: 2165419243

-----------------------------------------------------------------------------------------------------
| Id  | Operation                | Name                    | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |                         |  1094 | 55794 |     9  (23)| 00:00:01 |
|*  1 |  VIEW                    |                         |  1094 | 55794 |     9  (23)| 00:00:01 |
|*  2 |   WINDOW SORT PUSHED RANK|                         |  1094 | 55794 |     9  (23)| 00:00:01 |
|   3 |    HASH GROUP BY         |                         |  1094 | 55794 |     9  (23)| 00:00:01 |
|*  4 |     HASH JOIN            |                         |  1094 | 55794 |     7   (0)| 00:00:01 |
|   5 |      TABLE ACCESS FULL   | CITY_COUNTRY_REGION_DIM |   118 |  2950 |     3   (0)| 00:00:01 |
|   6 |      TABLE ACCESS FULL   | EMPLOYEE_FACT           |  1094 | 28444 |     4   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("DENSE_RANK"=1)
   2 - filter(DENSE_RANK() OVER ( PARTITION BY "EF"."YEAR_MONTH_ID" ORDER BY
              SUM("TOTAL_SALARY") DESC )<=1)
   4 - access("EF"."CITY_COUNTRY_REGION_ID"="C"."CITY_COUNTRY_REGION_ID")
```

(d) Task4 query tree



(e) New SQL

```
SELECT
    *
FROM
    (
        SELECT /*+ USE_MERGE(ef c) */
            ef.year_month_id,
            c.city,
            SUM(total_salary) AS total_salary,
            DENSE_RANK() OVER(PARTITION BY
                year_month_id
                ORDER BY
                    SUM(total_salary)
                DESC
            ) AS dense_rank
        FROM
```

```
            employee_fact ef,
            city_country_region_dim c
        WHERE
            ef.city_country_region_id = c.city_country_region_id
        GROUP BY
            c.city,
            ef.year_month_id
    )
WHERE
    dense_rank = 1;
```

(f) New query result

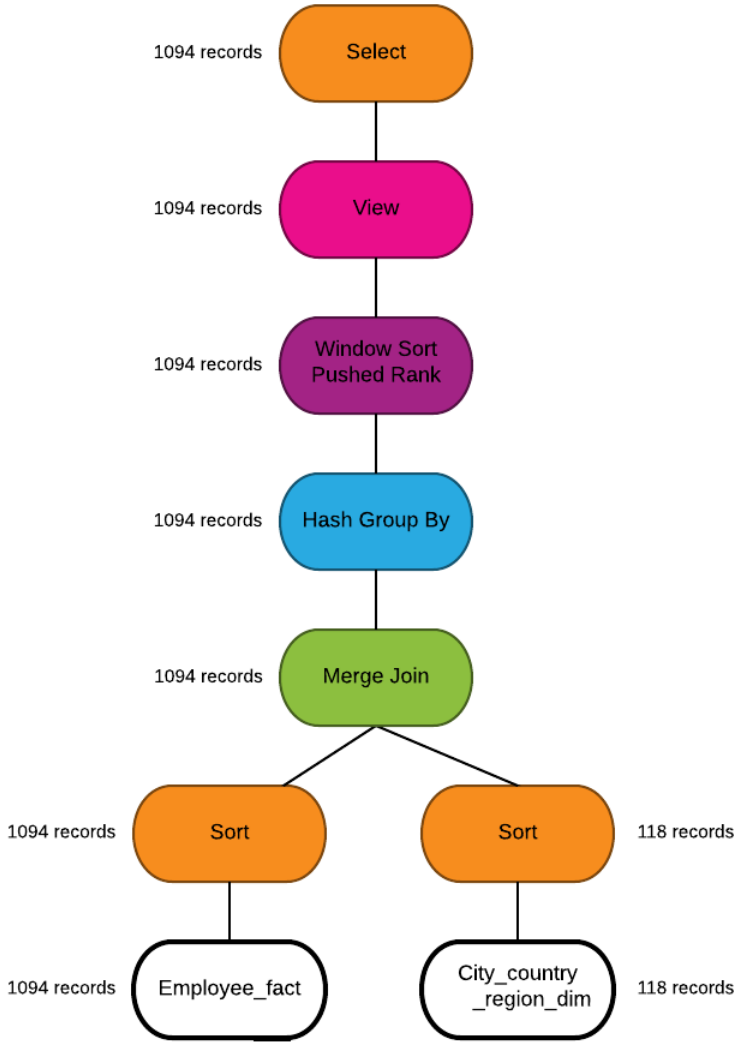
	YEAR_MONTH_ID	CITY	TOTAL_SALARY	DENSE_RANK
1	200401	Seattle	101816	1
2	200402	Seattle	101816	1
3	200403	Seattle	101816	1
4	200404	Seattle	101816	1
5	200405	Seattle	101816	1
6	200406	Seattle	101816	1
7	200407	Seattle	101816	1
8	200408	Seattle	101816	1
9	200409	Seattle	101816	1
10	200410	Seattle	101816	1
11	200411	Seattle	101816	1
12	200412	Seattle	101816	1
13	200501	Seattle	101816	1
14	200502	Seattle	101816	1
15	200503	Oxford	126800	1
16	200504	Oxford	126800	1
17	200505	Oxford	126800	1
18	200506	Oxford	126800	1
19	200507	Oxford	126800	1
20	200508	Oxford	135800	1
21	200509	Seattle	137516	1
22	200510	Seattle	137516	1
23	200511	Oxford	146300	1
24	200512	Oxford	153800	1
25	200601	Oxford	163400	1
26	200602	Oxford	163400	1
27	200603	Oxford	190000	1
28	200604	Oxford	198400	1
29	200605	Oxford	198400	1
30	200606	Oxford	198400	1
31	200607	Oxford	198400	1
32	200608	Oxford	198400	1
33	200609	Oxford	198400	1
34	200610	Oxford	198400	1

(g) New query execution plan

PLAN_TABLE_OUTPUT

```
Plan hash value: 843641112

-------------------------------------------------------------------------------------------------------
| Id  | Operation                | Name                      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |                           |  1094 | 55794 |    11  (37)| 00:00:01 |
|*  1 |  VIEW                    |                           |  1094 | 55794 |    11  (37)| 00:00:01 |
|*  2 |   WINDOW SORT PUSHED RANK|                           |  1094 | 55794 |    11  (37)| 00:00:01 |
|   3 |    HASH GROUP BY         |                           |  1094 | 55794 |    11  (37)| 00:00:01 |
|   4 |     MERGE JOIN           |                           |  1094 | 55794 |     9  (23)| 00:00:01 |
|   5 |      SORT JOIN           |                           |   118 |  2950 |     4  (25)| 00:00:01 |
|   6 |       TABLE ACCESS FULL  | CITY_COUNTRY_REGION_DIM   |   118 |  2950 |     3   (0)| 00:00:01 |
|*  7 |      SORT JOIN           |                           |  1094 | 28444 |     5  (20)| 00:00:01 |
|   8 |       TABLE ACCESS FULL  | EMPLOYEE_FACT             |  1094 | 28444 |     4   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("DENSE_RANK"=1)
   2 - filter(DENSE_RANK() OVER ( PARTITION BY "EF"."YEAR_MONTH_ID" ORDER BY
              SUM("TOTAL_SALARY") DESC )<=1)
   7 - access("EF"."CITY_COUNTRY_REGION_ID"="C"."CITY_COUNTRY_REGION_ID")
       filter("EF"."CITY_COUNTRY_REGION_ID"="C"."CITY_COUNTRY_REGION_ID")
```

(h) New query tree



(i) Comparison

	Task4 query	New query
Execution time	6	8
Which is better	√	
Join Type	Hash Join	Merge join
Reason	In task4 query, it use hash join, which pick a smaller table and push records into a hash table, and then put records in the lager table to the hash table to match the key, which save a lot of time than merge join, and more efficient.	

Query 6

Show ascending dense rank of total sales by each city partition by season.

(a) SQL from task 4

```
SELECT
    s.season_desc AS season,
    c.city,
    SUM(sf.total_sales) AS total_sales,
    DENSE_RANK() OVER(PARTITION BY
        s.season_desc
        ORDER BY
            SUM(sf.total_sales)
    ) AS dense_rank
FROM
    sales_order_fact sf,
    season_dim s,
    city_country_region_dim c
WHERE
        sf.season_id = s.season_id
    AND
        c.city_country_region_id = sf.city_country_region_id
GROUP BY
    s.season_desc,
    c.city;
```

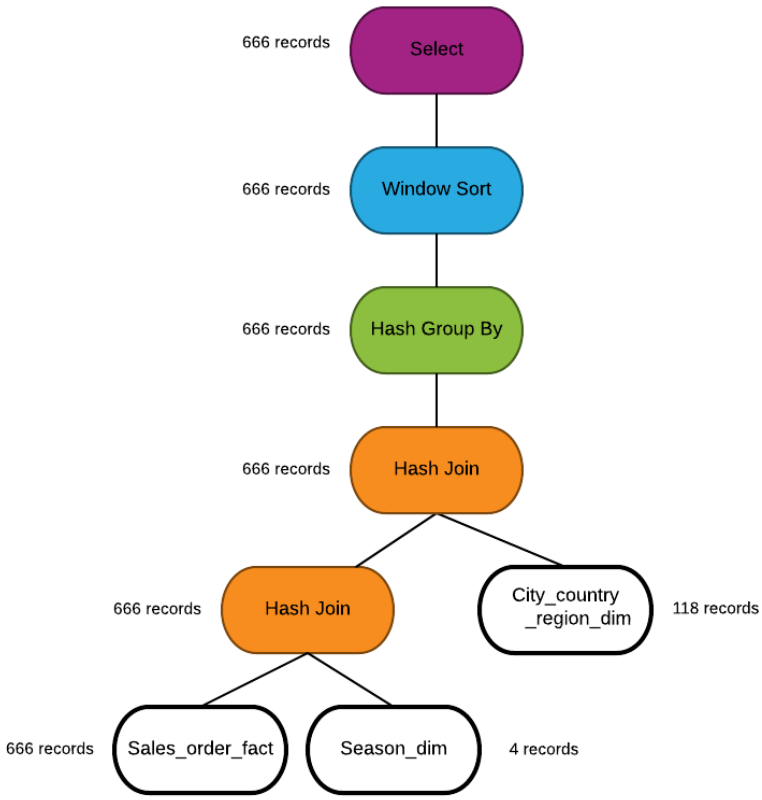
(b) Task4 query result

	SEASON	CITY	TOTAL_SALES	DENSE_RANK
1	Autumn	Bloomington	2273.6	1
2	Autumn	Dubuque	25691.3	2
3	Autumn	Des Moines	27249.6	3
4	Autumn	Elkhart	31630	4
5	Autumn	Eau Claire	39684.7	5
6	Autumn	Indianapolis	40727.4	6
7	Autumn	Kokomo	48154.6	7
8	Autumn	Milwaukee	49799.3	8
9	Autumn	Madison	52758.9	9
10	Autumn	Cedar Rapids	71784.9	10
11	Autumn	Grand Rapids	79498.6	11
12	Autumn	South Bend	100081.2	12
13	Autumn	Minneapolis	128322.8	13
14	Spring	Elkhart	990.4	1
15	Spring	Detroit	19441.44	2
16	Spring	Bloomington	20991.2	3
17	Spring	Kokomo	29669.9	4
18	Spring	Milwaukee	30051.9	5
19	Spring	Eau Claire	51329.7	6
20	Spring	Grand Rapids	71064.34	7
21	Spring	Cedar Rapids	72736	8
22	Spring	South Bend	92661.4	9
23	Spring	Indianapolis	186804	10
24	Spring	Madison	296657.8	11
25	Summer	Cedar Rapids	7824.67	1
26	Summer	Saginaw	16361.4	2
27	Summer	South Bend	20940	3
28	Summer	Bloomington	37553.8	4
29	Summer	Madison	69992.15	5
30	Summer	Lansing	109151.1	6
31	Summer	Indianapolis	165845.4	7
32	Summer	Milwaukee	337563.67	8
33	Winter	Minneapolis	288	1
34	Winter	Green Bay	640	2
35	Winter	Wausau	977.2	3

(c) Task 4 query execution plan

```
PLAN_TABLE_OUTPUT
Plan hash value: 4153508684

-----------------------------------------------------------------------------------------------
| Id  | Operation            | Name                  | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |                       |   116 |  8004 |    12  (17)| 00:00:01 |
|   1 |  WINDOW SORT         |                       |   116 |  8004 |    12  (17)| 00:00:01 |
|   2 |   HASH GROUP BY      |                       |   116 |  8004 |    12  (17)| 00:00:01 |
|*  3 |    HASH JOIN         |                       |   666 | 45954 |    10   (0)| 00:00:01 |
|   4 |     TABLE ACCESS FULL | CITY_COUNTRY_REGION_DIM |   118 |  2950 |     3   (0)| 00:00:01 |
|*  5 |     HASH JOIN        |                       |   666 | 29304 |     7   (0)| 00:00:01 |
|   6 |      TABLE ACCESS FULL| SEASON_DIM            |     4 |    80 |     3   (0)| 00:00:01 |
|   7 |      TABLE ACCESS FULL| SALES_ORDER_FACT      |   666 | 15984 |     4   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("C"."CITY_COUNTRY_REGION_ID"="SF"."CITY_COUNTRY_REGION_ID")
   5 - access("SF"."SEASON_ID"="S"."SEASON_ID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

(d) Task4 query tree



(e) New SQL

```
SELECT
    s.season_desc AS season,
    c.city,
    SUM(sf.total_sales) AS total_sales,
    DENSE_RANK() OVER(PARTITION BY
        s.season_desc
        ORDER BY
            SUM(sf.total_sales)
    ) AS dense_rank
FROM
    sales_order_fact sf,
    season_dim s,
    city_country_region_dim c
WHERE
        sf.season_id = s.season_id
    AND
        c.city_country_region_id = sf.city_country_region_id
GROUP BY
    s.season_desc,
    c.city
ORDER BY
    s.season_desc,
    c.city;
```

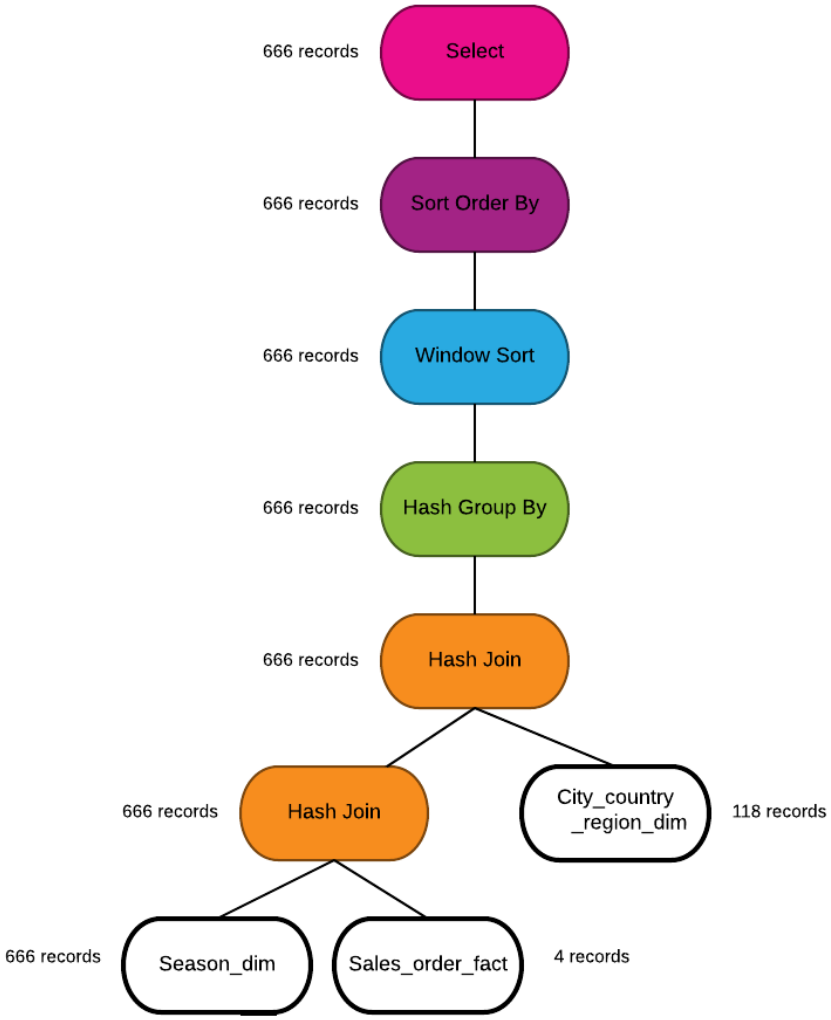
(f) New query result

	SEASON	CITY	TOTAL_SALES	DENSE_RANK
1	Autumn	Bloomington	2273.6	1
2	Autumn	Cedar Rapids	71784.9	10
3	Autumn	Des Moines	27249.6	3
4	Autumn	Dubuque	25691.3	2
5	Autumn	Eau Claire	39684.7	5
6	Autumn	Elkhart	31630	4
7	Autumn	Grand Rapids	79498.6	11
8	Autumn	Indianapolis	40727.4	6
9	Autumn	Kokomo	48154.6	7
10	Autumn	Madison	52758.9	9
11	Autumn	Milwaukee	49799.3	8
12	Autumn	Minneapolis	128322.8	13
13	Autumn	South Bend	100081.2	12
14	Spring	Bloomington	20991.2	3
15	Spring	Cedar Rapids	72736	8
16	Spring	Detroit	19441.44	2
17	Spring	Eau Claire	51329.7	6
18	Spring	Elkhart	990.4	1
19	Spring	Grand Rapids	71064.34	7
20	Spring	Indianapolis	186804	10
21	Spring	Kokomo	29669.9	4
22	Spring	Madison	296657.8	11
23	Spring	Milwaukee	30051.9	5
24	Spring	South Bend	92661.4	9
25	Summer	Bloomington	37553.8	4
26	Summer	Cedar Rapids	7824.67	1
27	Summer	Indianapolis	165845.4	7
28	Summer	Lansing	109151.1	6
29	Summer	Madison	69992.15	5
30	Summer	Milwaukee	337563.67	8
31	Summer	Saginaw	16361.4	2
32	Summer	South Bend	20940	3
33	Winter	Ann Arbor	9106.48	7
34	Winter	Beloit	26341	10

(g) New query execution plan

```
PLAN_TABLE_OUTPUT
Plan hash value: 3115110740

-----------------------------------------------------------------------------------------------------
| Id  | Operation              | Name                    | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |                         |   116 |  8004 |    13  (24)| 00:00:01 |
|   1 |  SORT ORDER BY         |                         |   116 |  8004 |    13  (24)| 00:00:01 |
|   2 |   WINDOW SORT          |                         |   116 |  8004 |    13  (24)| 00:00:01 |
|   3 |    HASH GROUP BY       |                         |   116 |  8004 |    13  (24)| 00:00:01 |
|*  4 |     HASH JOIN          |                         |   666 | 45954 |    10   (0)| 00:00:01 |
|   5 |      TABLE ACCESS FULL | CITY_COUNTRY_REGION_DIM |   118 |  2950 |     3   (0)| 00:00:01 |
|*  6 |      HASH JOIN         |                         |   666 | 29304 |     7   (0)| 00:00:01 |
|   7 |       TABLE ACCESS FULL| SEASON_DIM              |     4 |    80 |     3   (0)| 00:00:01 |
|   8 |       TABLE ACCESS FULL| SALES_ORDER_FACT        |   666 | 15984 |     4   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   4 - access("C"."CITY_COUNTRY_REGION_ID"="SF"."CITY_COUNTRY_REGION_ID")
   6 - access("SF"."SEASON_ID"="S"."SEASON_ID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

(h) New query tree



(i) Comparison

	Task4 query	New query
Execution time	7	8
Which is better	√	
Difference	No order by	Order by
Reason	In the new query, it use order by, but the results of two queries are totally the same, so it is not necessary and waste time.	

Query 7

Calculate each month sales and total sales in the latest 3 months of each month from 2004 Jan to 2008 Dec of each products from the listed three products ("KB 101/EN","LaserPro 600/6/BW","Screws <B.28.S>").

(a) SQL from task 4

```
SELECT
    a.product_id,
    a.year_month_id,
    SUM(nvl(sf.total_sales,0) ) AS month_sales,
    SUM(
        SUM(nvl(sf.total_sales,0) )
    ) OVER(PARTITION BY
        a.product_id
        ORDER BY
            a.product_id,
            a.year_month_id
        ROWS
            2
        PRECEDING
    ) AS moving_3_month_total_sales
FROM
    (
        SELECT DISTINCT
            sf2.product_id,
            y.year_month_id
        FROM
            sales_order_fact sf2,
            year_month_dim y,
            product_dim p
        WHERE
                y.year_month_id <= '200812'
            AND
                y.year_month_id >= '200401'
            AND
                p.product_id = sf2.product_id
            AND
                p.product_name IN (
                    'KB 101/EN','LaserPro 600/6/BW','Screws <B.28.S>'
                )
    ) a
```

```
    LEFT JOIN sales_order_fact sf ON
        sf.product_id = a.product_id
    AND
        sf.year_month_id = a.year_month_id
GROUP BY
    a.product_id,
    a.year_month_id;
```

(b) Task4 query result

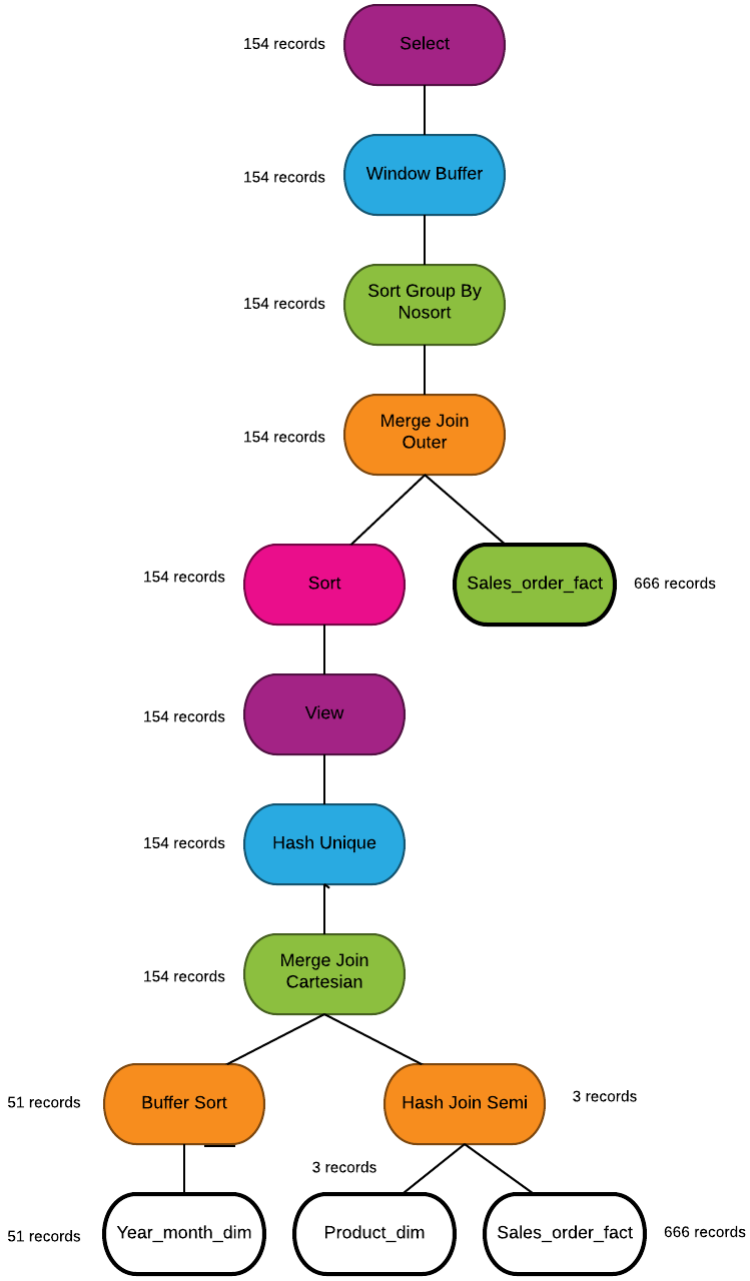
	PRODUCT_ID	YEAR_MONTH_ID	MONTH_SALES	MOVING_3_MONTH_TOTAL_SALES
1	3106	200401	0	0
2	3106	200402	0	0
3	3106	200403	0	0
4	3106	200404	0	0
5	3106	200405	0	0
6	3106	200406	0	0
7	3106	200407	0	0
8	3106	200408	0	0
9	3106	200409	0	0
10	3106	200410	0	0
11	3106	200411	0	0
12	3106	200412	0	0
13	3106	200501	0	0
14	3106	200502	0	0
15	3106	200503	0	0
16	3106	200504	0	0
17	3106	200505	0	0
18	3106	200506	0	0
19	3106	200507	0	0
20	3106	200508	0	0
21	3106	200509	0	0
22	3106	200510	0	0
23	3106	200511	0	0
24	3106	200512	0	0
25	3106	200601	0	0
26	3106	200602	7200	7200
27	3106	200603	8160	15360
28	3106	200604	0	15360
29	3106	200605	0	8160
30	3106	200606	0	0
31	3106	200607	144	144
32	3106	200608	0	144
33	3106	200609	48	192
34	3106	200610	0	48

(c) Task 4 query execution plan

PLAN_TABLE_OUTPUT

```
Plan hash value: 2035182304

-------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name              | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |                   |   154 |  4158 |     19  (11)| 00:00:01 |
|   1 |  WINDOW BUFFER             |                   |   154 |  4158 |     19  (11)| 00:00:01 |
|   2 |   SORT GROUP BY NOSORT     |                   |   154 |  4158 |     19  (11)| 00:00:01 |
|   3 |    MERGE JOIN OUTER        |                   |   154 |  4158 |     19  (11)| 00:00:01 |
|   4 |     SORT JOIN              |                   |   154 |  1694 |     14   (8)| 00:00:01 |
|   5 |      VIEW                  |                   |   154 |  1694 |     14   (8)| 00:00:01 |
|   6 |       HASH UNIQUE          |                   |   154 |  4774 |     14   (8)| 00:00:01 |
|   7 |        MERGE JOIN CARTESIAN|                   |   154 |  4774 |     13   (0)| 00:00:01 |
|*  8 |         HASH JOIN SEMI     |                   |     3 |    72 |      8   (0)| 00:00:01 |
|*  9 |          TABLE ACCESS FULL | PRODUCT_DIM       |     3 |    60 |      4   (0)| 00:00:01 |
|  10 |          TABLE ACCESS FULL | SALES_ORDER_FACT  |   666 |  2664 |      4   (0)| 00:00:01 |
|  11 |         BUFFER SORT        |                   |    51 |   357 |      9   (0)| 00:00:01 |
|* 12 |          TABLE ACCESS FULL | YEAR_MONTH_DIM    |    51 |   357 |      2   (0)| 00:00:01 |
|* 13 |     SORT JOIN              |                   |   666 | 10656 |      5  (20)| 00:00:01 |
|  14 |      TABLE ACCESS FULL     | SALES_ORDER_FACT  |   666 | 10656 |      4   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   8 - access("P"."PRODUCT_ID"="SF2"."PRODUCT_ID")
   9 - filter("P"."PRODUCT_NAME"='KB 101/EN' OR "P"."PRODUCT_NAME"='LaserPro 600/6/BW'
              OR "P"."PRODUCT_NAME"='Screws <B.28.S>')
  12 - filter("Y"."YEAR_MONTH_ID"<='200812' AND "Y"."YEAR_MONTH_ID">='200401')
  13 - access("SF"."PRODUCT_ID"(+)="A"."PRODUCT_ID" AND
              "SF"."YEAR_MONTH_ID"(+)="A"."YEAR_MONTH_ID")
       filter("SF"."YEAR_MONTH_ID"(+)="A"."YEAR_MONTH_ID" AND
              "SF"."PRODUCT_ID"(+)="A"."PRODUCT_ID")
```

(d) Task4 query tree



(e) New SQL

```
CREATE INDEX product_id_idx ON
    sales_order_fact ( product_id );

SELECT
    a.product_id,
    a.year_month_id,
    SUM(nvl(sf.total_sales,0) ) AS month_sales,
```

```
    SUM(
        SUM(nvl(sf.total_sales,0) )
    ) OVER(PARTITION BY
        a.product_id
        ORDER BY
            a.product_id,
            a.year_month_id
        ROWS
            2
        PRECEDING
    ) AS moving_3_month_total_sales
FROM
    (
        SELECT DISTINCT
            sf2.product_id,
            y.year_month_id
        FROM
            sales_order_fact sf2,
            year_month_dim y,
            product_dim p
        WHERE
                y.year_month_id <= '200812'
            AND
                y.year_month_id >= '200401'
            AND
                p.product_id = sf2.product_id
            AND
                p.product_name IN (
                    'KB 101/EN','LaserPro 600/6/BW','Screws <B.28.S>'
                )
    ) a
    LEFT JOIN sales_order_fact sf ON
        sf.product_id = a.product_id
    AND
        sf.year_month_id = a.year_month_id
GROUP BY
    a.product_id,
    a.year_month_id;
```

(f) New query result

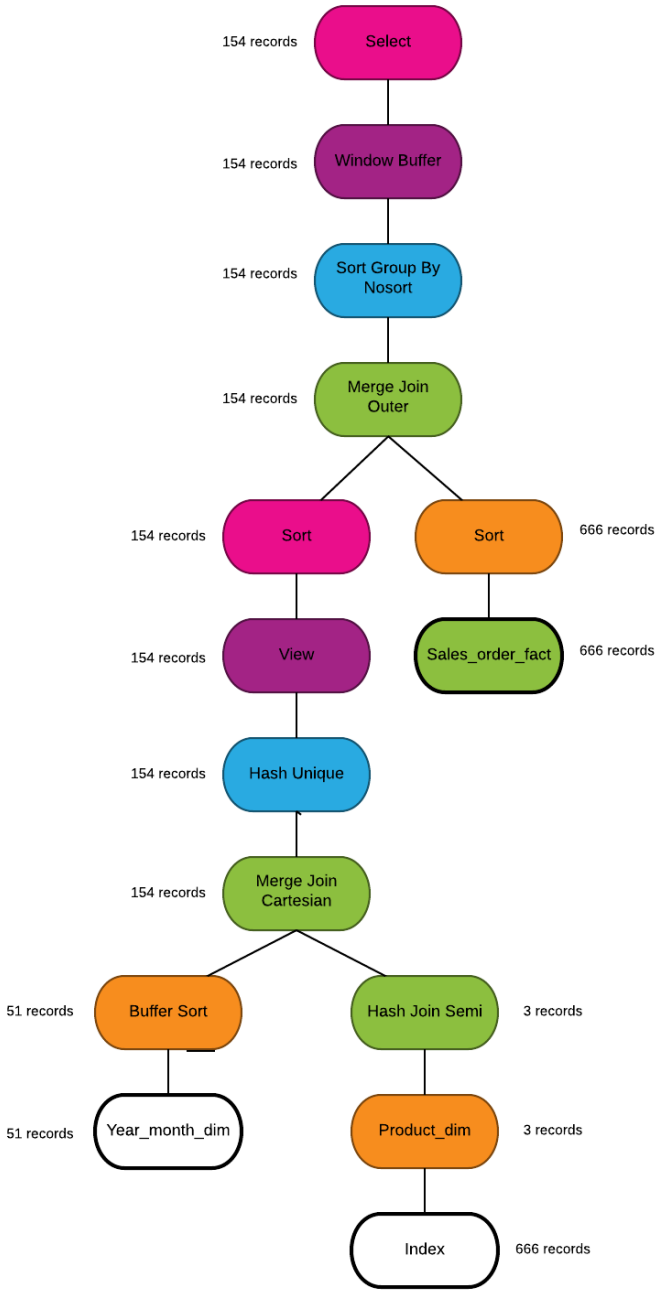
	PRODUCT_ID	YEAR_MONTH_ID	MONTH_SALES	MOVING_3_MONTH_TOTAL_SALES
1	3106	200401	0	0
2	3106	200402	0	0
3	3106	200403	0	0
4	3106	200404	0	0
5	3106	200405	0	0
6	3106	200406	0	0
7	3106	200407	0	0
8	3106	200408	0	0
9	3106	200409	0	0
10	3106	200410	0	0
11	3106	200411	0	0
12	3106	200412	0	0
13	3106	200501	0	0
14	3106	200502	0	0
15	3106	200503	0	0
16	3106	200504	0	0
17	3106	200505	0	0
18	3106	200506	0	0
19	3106	200507	0	0
20	3106	200508	0	0
21	3106	200509	0	0
22	3106	200510	0	0
23	3106	200511	0	0
24	3106	200512	0	0
25	3106	200601	0	0
26	3106	200602	7200	7200
27	3106	200603	8160	15360
28	3106	200604	0	15360
29	3106	200605	0	8160
30	3106	200606	0	0
31	3106	200607	144	144
32	3106	200608	0	144
33	3106	200609	48	192
34	3106	200610	0	48

(g) New query execution plan

PLAN_TABLE_OUTPUT

```
Plan hash value: 2750233047

-------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name              | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                   |   154 |  4158 |    17  (12)| 00:00:01 |
|   1 |  WINDOW BUFFER                |                   |   154 |  4158 |    17  (12)| 00:00:01 |
|   2 |   SORT GROUP BY NOSORT        |                   |   154 |  4158 |    17  (12)| 00:00:01 |
|   3 |    MERGE JOIN OUTER           |                   |   154 |  4158 |    17  (12)| 00:00:01 |
|   4 |     SORT JOIN                 |                   |   154 |  1694 |    12   (9)| 00:00:01 |
|   5 |      VIEW                     |                   |   154 |  1694 |    12   (9)| 00:00:01 |
|   6 |       HASH UNIQUE             |                   |   154 |  4774 |    12   (9)| 00:00:01 |
|   7 |        MERGE JOIN CARTESIAN   |                   |   154 |  4774 |    11   (0)| 00:00:01 |
|*  8 |         HASH JOIN SEMI        |                   |     3 |    72 |     6   (0)| 00:00:01 |
|*  9 |          TABLE ACCESS FULL    | PRODUCT_DIM       |     3 |    60 |     4   (0)| 00:00:01 |
|  10 |          INDEX FAST FULL SCAN | PRODUCT_ID_IDX    |   666 |  2664 |     2   (0)| 00:00:01 |
|  11 |         BUFFER SORT           |                   |    51 |   357 |     9   (0)| 00:00:01 |
|* 12 |          TABLE ACCESS FULL    | YEAR_MONTH_DIM    |    51 |   357 |     2   (0)| 00:00:01 |
|* 13 |       SORT JOIN               |                   |   666 | 10656 |     5  (20)| 00:00:01 |
|  14 |        TABLE ACCESS FULL      | SALES_ORDER_FACT  |   666 | 10656 |     4   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   8 - access("P"."PRODUCT_ID"="SF2"."PRODUCT_ID")
   9 - filter("P"."PRODUCT_NAME"='KB 101/EN' OR "P"."PRODUCT_NAME"='LaserPro 600/6/BW'
              OR "P"."PRODUCT_NAME"='Screws <B.28.S>')
  12 - filter("Y"."YEAR_MONTH_ID"<='200812' AND "Y"."YEAR_MONTH_ID">='200401')
  13 - access("SF"."PRODUCT_ID"(+)="A"."PRODUCT_ID" AND
              "SF"."YEAR_MONTH_ID"(+)="A"."YEAR_MONTH_ID")
       filter("SF"."YEAR_MONTH_ID"(+)="A"."YEAR_MONTH_ID" AND
              "SF"."PRODUCT_ID"(+)="A"."PRODUCT_ID")
```

(h) New query tree



(i) Comparison

	Task4 query	New query
Execution time	14	14
Which is better		√
Difference	No index	Use index
Reason	In the new query, an index on product_id in product_dim is used, it use index instead of table full scan, which save storage and is more efficient.	

Query 8

Calculate each year salary and cumulative total salary in the before years of each year from 2004 to 2008.

(a) SQL from task 4

```
SELECT
    c.country_name,
    ym.year,
    SUM(ef.total_salary) AS year_salary,
    SUM(
        SUM(ef.total_salary)
    ) OVER(PARTITION BY
        c.country_name
        ORDER BY
            c.country_name,
            ym.year
        ROWS UNBOUNDED PRECEDING
    ) AS cumulative_total_salary
FROM
    employee_fact ef,
    year_month_dim ym,
    city_country_region_dim c
WHERE
        ef.year_month_id = ym.year_month_id
    AND
        ef.city_country_region_id = c.city_country_region_id
GROUP BY
    c.country_name,
    ym.year;
```

(b) Task4 query result

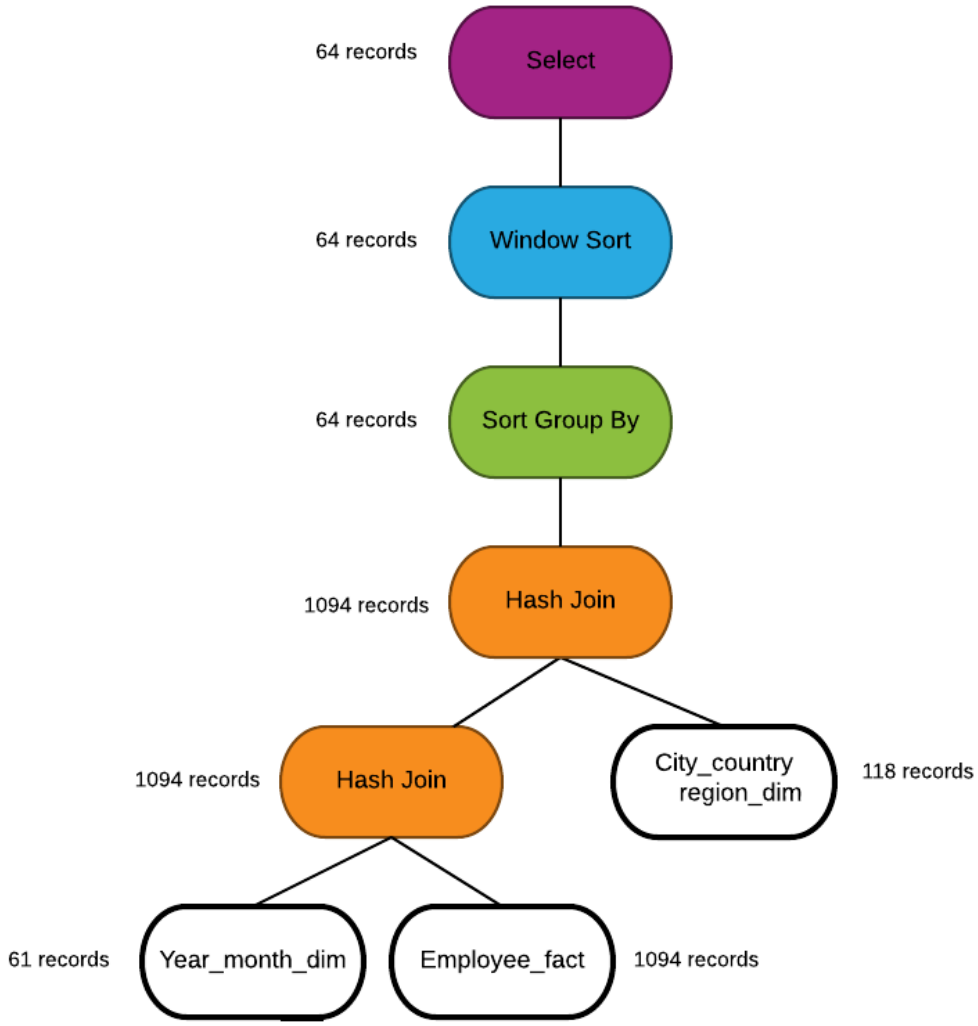
	COUNTRY_NAME	YEAR	YEAR_SALARY	CUMULATIVE_TOTAL_SALARY
1	Canada	2004	143000	143000
2	Canada	2005	186000	329000
3	Canada	2006	228000	557000
4	Canada	2007	228000	785000
5	Canada	2008	228000	1013000
6	Germany	2004	120000	120000
7	Germany	2005	120000	240000
8	Germany	2006	120000	360000
9	Germany	2007	120000	480000
10	Germany	2008	120000	600000
11	United Kingdom	2004	468000	468000
12	United Kingdom	2005	1573500	2041500
13	United Kingdom	2006	2401900	4443400
14	United Kingdom	2007	2985200	7428600
15	United Kingdom	2008	3759500	11188100
16	United States of America	2004	1567292	1567292
17	United States of America	2005	2161092	3728384
18	United States of America	2006	3243892	6972276
19	United States of America	2007	3715092	10687368
20	United States of America	2008	4097192	14784560

(c) Task 4 query execution plan

PLAN_TABLE_OUTPUT

```
Plan hash value: 3260075010

-------------------------------------------------------------------------------------------------
| Id  | Operation             | Name                    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |                         |    64 |  4544 |    11  (10)| 00:00:01 |
|   1 |  WINDOW NOSORT        |                         |    64 |  4544 |    11  (10)| 00:00:01 |
|   2 |   SORT GROUP BY       |                         |    64 |  4544 |    11  (10)| 00:00:01 |
|*  3 |    HASH JOIN          |                         |  1094 | 77674 |    10   (0)| 00:00:01 |
|   4 |     TABLE ACCESS FULL | CITY_COUNTRY_REGION_DIM |   118 |  3894 |     3   (0)| 00:00:01 |
|*  5 |     HASH JOIN         |                         |  1094 | 41572 |     7   (0)| 00:00:01 |
|   6 |      TABLE ACCESS FULL| YEAR_MONTH_DIM          |    61 |   732 |     3   (0)| 00:00:01 |
|   7 |      TABLE ACCESS FULL| EMPLOYEE_FACT           |  1094 | 28444 |     4   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("EF"."CITY_COUNTRY_REGION_ID"="C"."CITY_COUNTRY_REGION_ID")
   5 - access("EF"."YEAR_MONTH_ID"="YM"."YEAR_MONTH_ID")
```

(d) Task4 query tree



(e) New SQL

```
SELECT /*+ use_merge (ym c)*/
    innerquery.country_name,
    innerquery.year,
    SUM(
        innerquery.year_salary
    ) OVER(PARTITION BY
        innerquery.country_name
        ORDER BY
            country_name,
            year
        ROWS UNBOUNDED PRECEDING
    ) AS cumulative_total_salary
```

```
FROM
    (
        SELECT /*+ no_merge*/
            c.country_name,
            ym.year,
            SUM(ef.total_salary) AS year_salary
        FROM
            employee_fact ef,
            year_month_dim ym,
            city_country_region_dim c
        WHERE
                ef.year_month_id = ym.year_month_id
            AND
                ef.city_country_region_id = c.city_country_region_id
        GROUP BY
            c.country_name,
            ym.year
    ) innerquery;
```

(f) New query result

	COUNTRY_NAME	YEAR	YEAR_SALARY	CUMULATIVE_TOTAL_SALARY
1	Canada	2004	143000	143000
2	Canada	2005	186000	329000
3	Canada	2006	228000	557000
4	Canada	2007	228000	785000
5	Canada	2008	228000	1013000
6	Germany	2004	120000	120000
7	Germany	2005	120000	240000
8	Germany	2006	120000	360000
9	Germany	2007	120000	480000
10	Germany	2008	120000	600000
11	United Kingdom	2004	468000	468000
12	United Kingdom	2005	1573500	2041500
13	United Kingdom	2006	2401900	4443400
14	United Kingdom	2007	2985200	7428600
15	United Kingdom	2008	3759500	11188100
16	United States of America	2004	1567292	1567292
17	United States of America	2005	2161092	3728384
18	United States of America	2006	3243892	6972276
19	United States of America	2007	3715092	10687368
20	United States of America	2008	4097192	14784560

(g) New query execution plan

PLAN_TABLE_OUTPUT

```
Plan hash value: 2344766503

-------------------------------------------------------------------------------------------------------
| Id  | Operation              | Name                    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |                         |    64 |  2304 |    11  (10)| 00:00:01 |
|   1 |  WINDOW SORT           |                         |    64 |  2304 |    11  (10)| 00:00:01 |
|   2 |   VIEW                 |                         |    64 |  2304 |    11  (10)| 00:00:01 |
|   3 |    HASH GROUP BY       |                         |    64 |  4544 |    11  (10)| 00:00:01 |
|*  4 |     HASH JOIN          |                         |  1094 | 77674 |    10   (0)| 00:00:01 |
|   5 |      TABLE ACCESS FULL | CITY_COUNTRY_REGION_DIM |   118 |  3894 |     3   (0)| 00:00:01 |
|*  6 |      HASH JOIN         |                         |  1094 | 41572 |     7   (0)| 00:00:01 |
|   7 |       TABLE ACCESS FULL| YEAR_MONTH_DIM          |    61 |   732 |     3   (0)| 00:00:01 |
|   8 |       TABLE ACCESS FULL| EMPLOYEE_FACT           |  1094 | 28444 |     4   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   4 - access("EF"."CITY_COUNTRY_REGION_ID"="C"."CITY_COUNTRY_REGION_ID")
   6 - access("EF"."YEAR_MONTH_ID"="YM"."YEAR_MONTH_ID")
```

(h) New query tree

(i) Comparison

	Task4 query	New query
Execution time	7	8
Which is better	√	
Difference	Sort group by	Hash group by, no merge
Reason	It looks that the task4 query is almost the same as the new query. In task4 query, it uses sort group by, which sort all the rows and group it. Meanwhile in new query, it uses hash group by, which save a lot of storage and more efficient.	