MONASH University

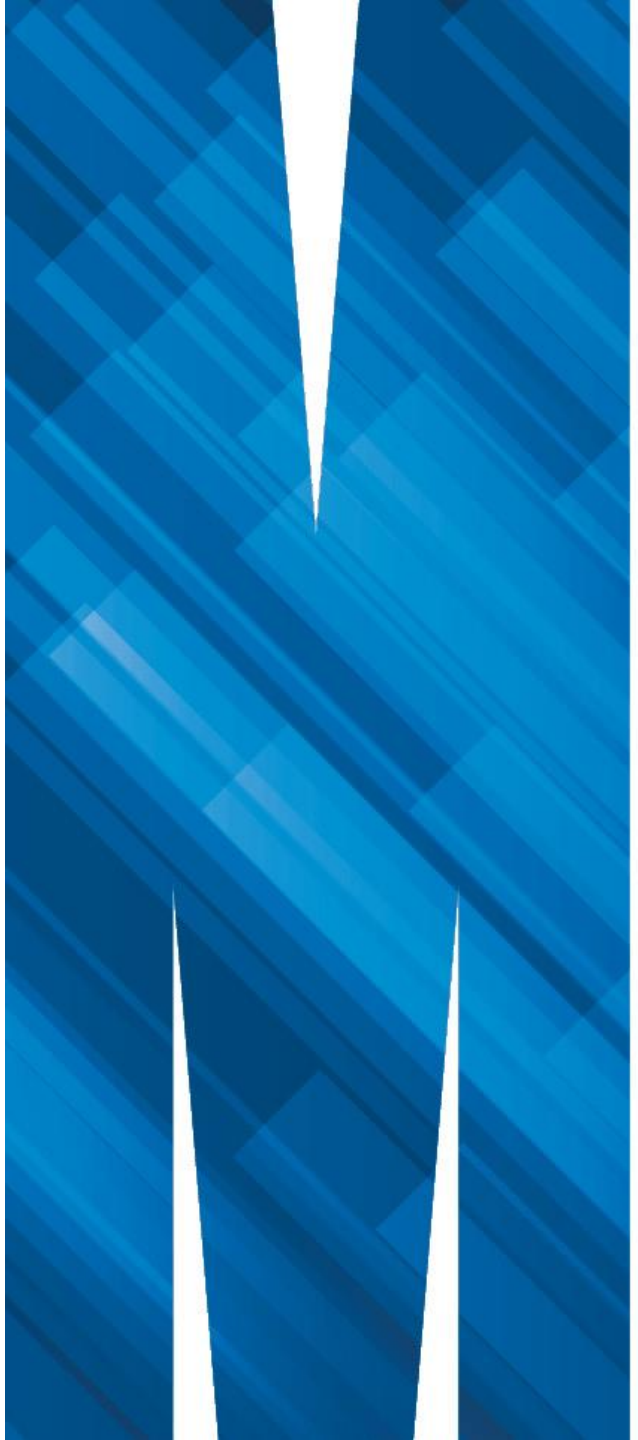
MONASH
INFORMATION
TECHNOLOGY

FIT3003 – Business Intelligence and Data Warehousing

Week 8 – OLAP

Semester 2, 2022

Developed by:
Dr. Agnes Haryanto
Agnes.Haryanto@monash.edu



Agenda

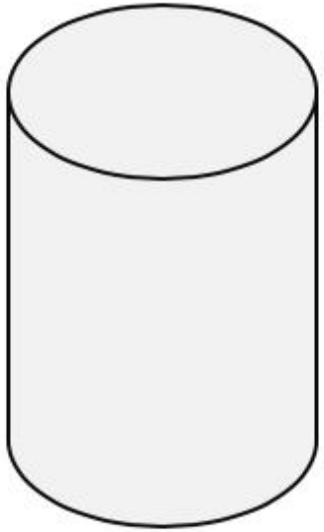
1. OLAP: An Overview
2. OLAP Queries
    1. Basic Aggregate Queries (Revision)
    2. Cube and Rollup
    3. Partial Cube and Partial Rollup
    4. Advanced Analysis
3. Business Intelligence Reporting

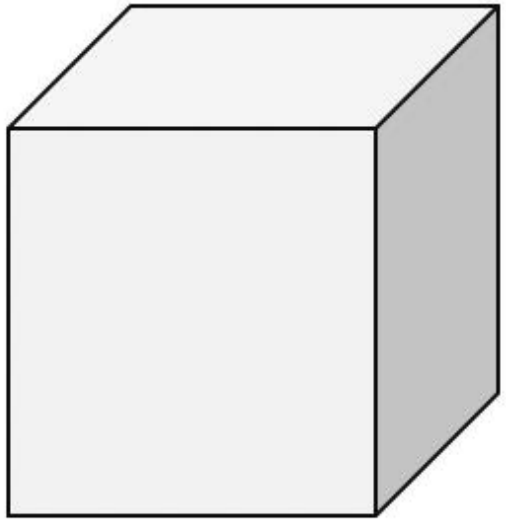
OLAP: An Overview

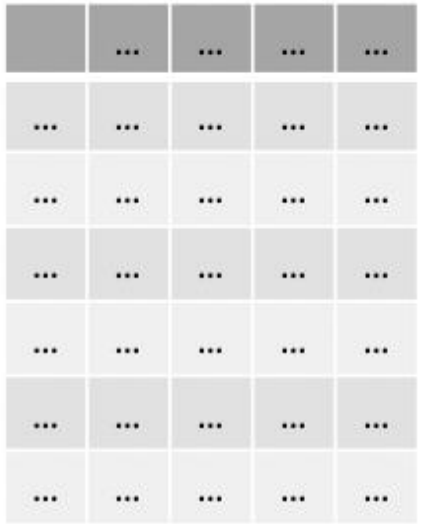
MONASH University

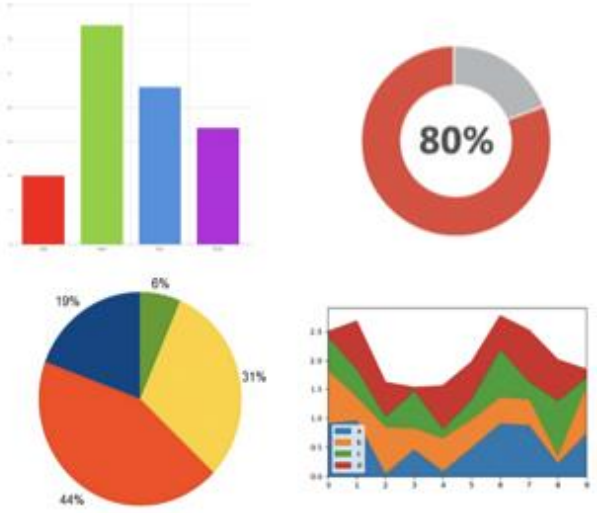
Recall – The Big Picture

Operational Database → Data Warehouse → OLAP → Business Intelligence



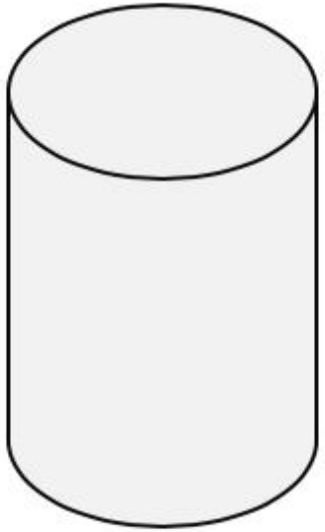


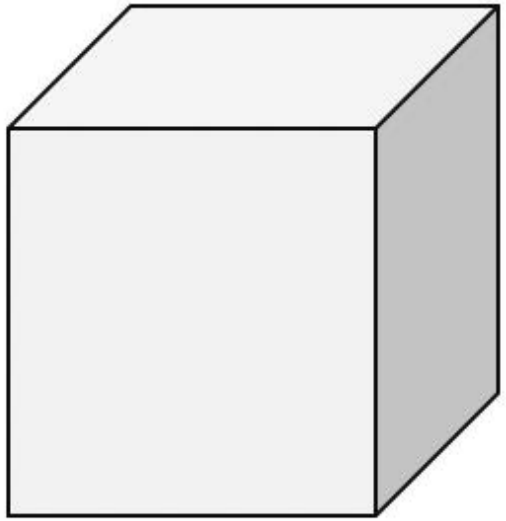


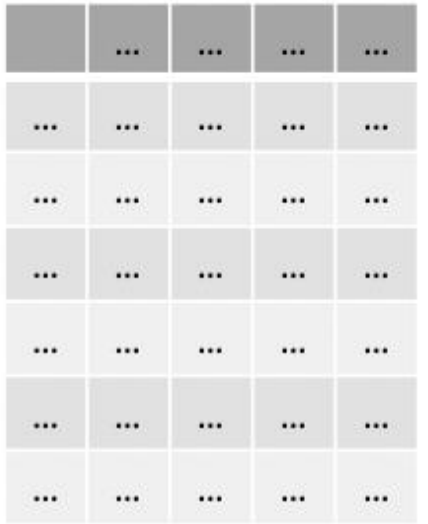


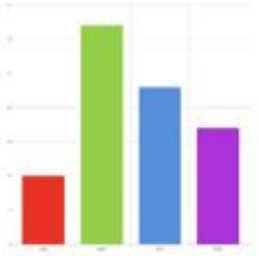
Recall – The Big Picture

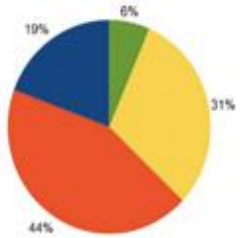
Operational Database → Data Warehouse → OLAP → Business Intelligence

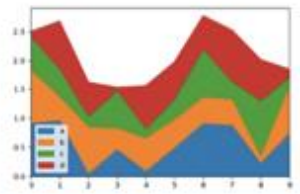






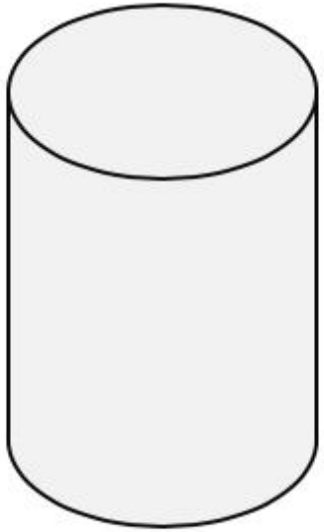


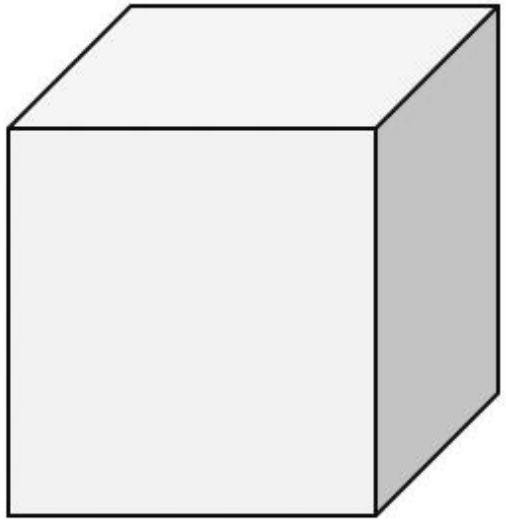


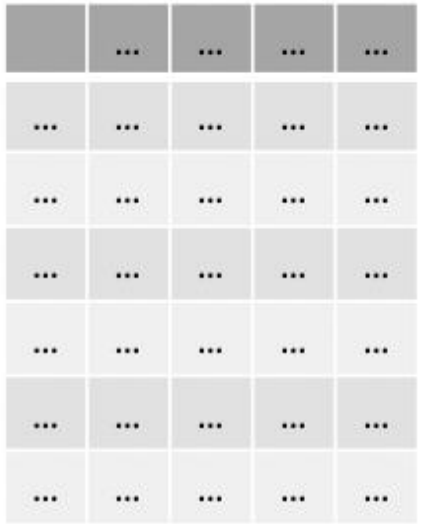


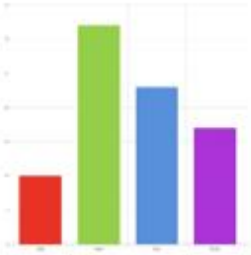
Recall – The Big Picture

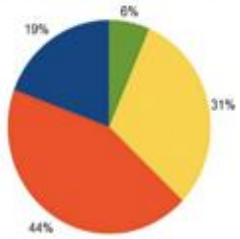
Operational Database → Data Warehouse → OLAP → Business Intelligence

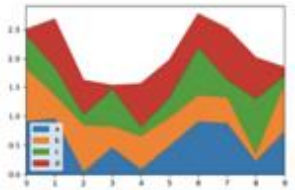






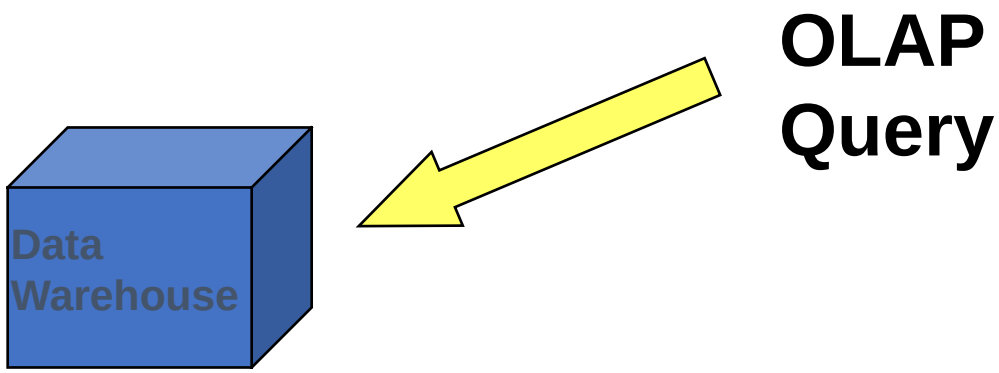




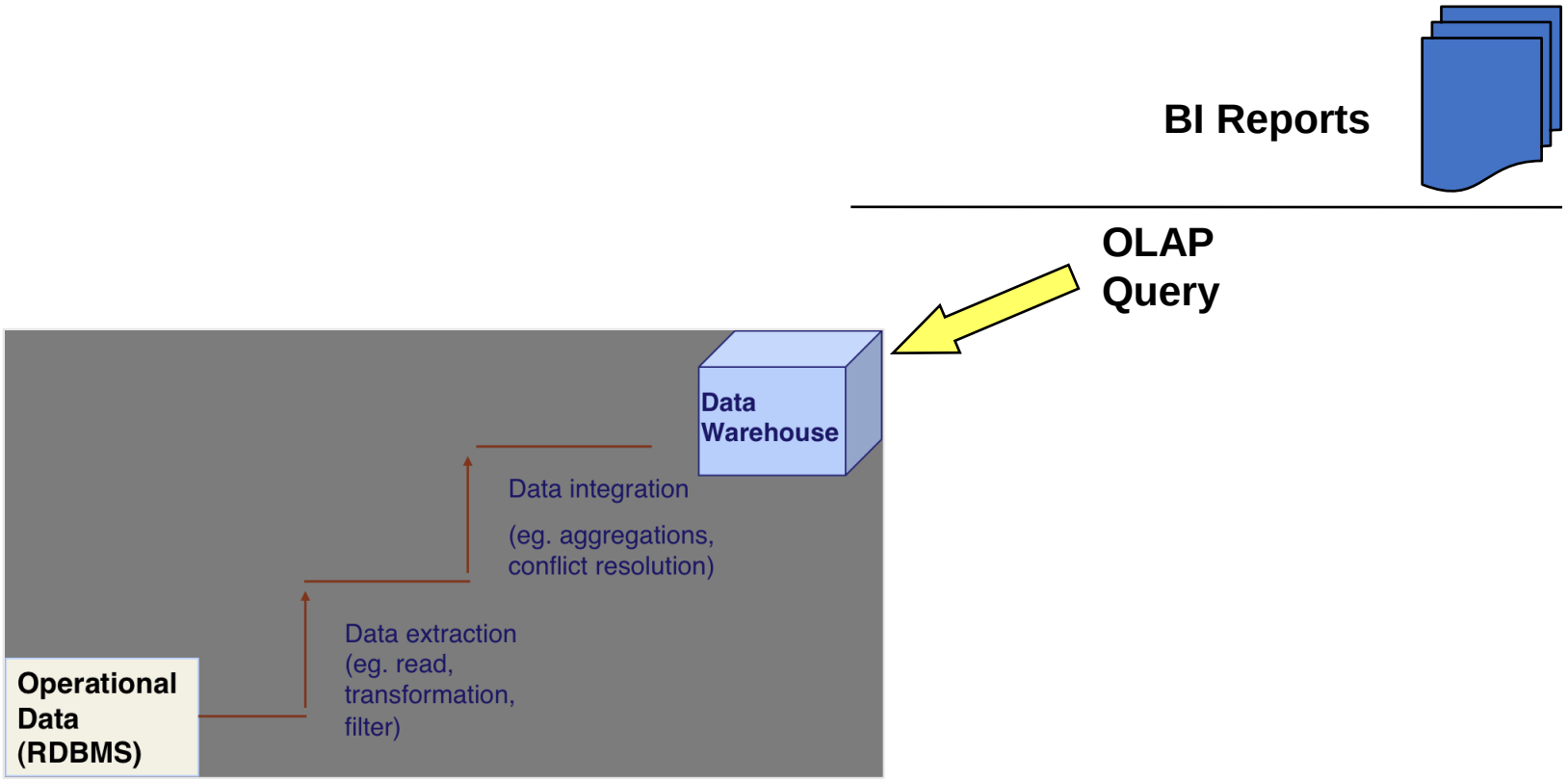


On-Line Analytical Processing (OLAP)

- The nature of the new Data Warehouse storage structure requires a tool that supports the retrieval of large number of records from very large data sets and summarizes them "on the fly" → OLAP (**On-Line Analytical Processing**) tool.



On-Line Analytical Processing (OLAP)



OLAP (On-Line Analytical Processing): Data processing that requires complex queries which typically involve **group-by** and **aggregate** operators.

OLAP Queries:

Basic Aggregate Queries

MONASH University

Basic AGGREGATE Functions (Revision)

Aggregate operations for computation purposes:

- **COUNT([distinct] A):** The number of (unique) values in the A column.
- **SUM([distinct] A):** The sum of all (unique) values in the A column.
- **AVG([distinct] A):** The average of all (unique) values in the A column.
- **MAX(A):** The maximum value in the A column.
- **MIN(A):** The minimum value in the A column.

Basic AGGREGATE Functions (Revision)

- COUNT: returns the number of tuples, which meet the specified condition.

```
SELECT COUNT(DISTINCT Dept) AS Num_Depts
FROM Subject;
```

- SUM: returns the sum of the values in a specified column (i.e. numeric column).

```
SELECT COUNT(*) AS hi_sal, SUM(salary)
FROM Lecturer
WHERE Salary > 4500
```

- MIN: returns the minimum value in a specified column (numeric or character).
- MAX: returns the maximum value in a specified column (numeric or character).
- AVG: returns the average of the values in a specified column (numeric or character).

Basic AGGREGATE Functions (Revision)

Use of GROUP BY changes the meaning of queries.

```
SELECT COUNT(SID)
FROM Enrol;
```

Count(SID)
3

```
SELECT Course, COUNT(SID)
FROM Enrol
GROUP BY Course;
```

Course	Count (SID)
101	2
113	1

OLAP Queries:

Cube and Rollup

MONASH University

Cube and Rollup

1. CUBE

   Extension to the GROUP BY clause to generate information in **cross-tabulation format** within a single query.

2. ROLLUP

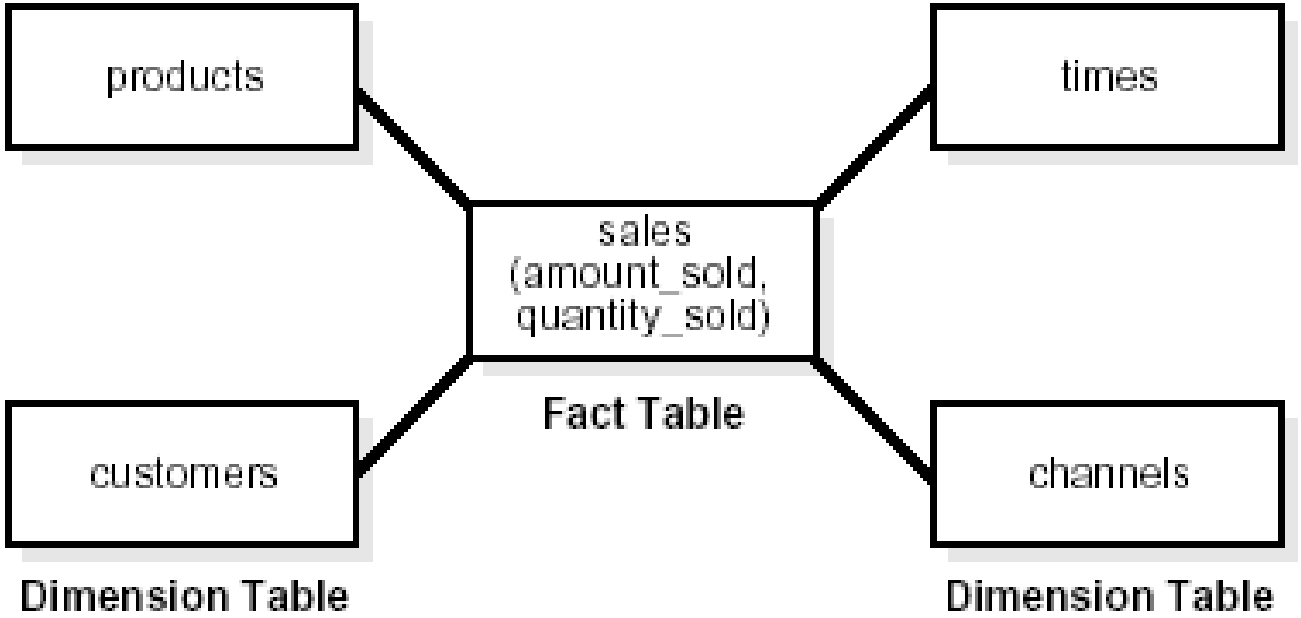
   Extension to the GROUP BY clause to generate aggregations **at increasing levels of granularity** from the most detailed to a grand total.

3. GROUPING

   To be used in conjunction with SELECT statement to **display information** about the aggregate levels and the relevant subtotals for each aggregate level.

Cube and Rollup

- Consider the following star schema example (source: Oracle 9i Data Warehousing Guide)



Cube and Rollup

- Consider the following example of cross-tabular report with subtotals based on the previous star schema (source: Oracle 9i Data Warehousing Guide). Note that country is an attribute from customer dimension.

Channel	Country		
	UK	US	Total
Direct Sales	1,378,126	2,835,557	4,213,683
Internet	911,739	1,732,240	2,643,979
Total	2,289,865	4,567,797	6,857,662

Cube: query and output

- **CUBE** gets input in a form of a set of attribute names to be grouped and it will produce *subtotals* for all the possible combinations of the specified attributes and the *grand total*.

Cube: query and output (example)

```
SELECT channel_desc, country_id, SUM(amount_sold)
as SALES$
FROM sales, customers, times, channels
WHERE sales.time_id = times.time_id
        AND sales.cust_id = customers.cust_id
        AND sales.channel_id =
            channels.channel_id
        AND channels.channel_desc IN
            ('Direct Sales', 'Internet')
        AND times.calendar_month_desc = '2000-09'
        AND country_id IN ('UK', 'US')
GROUP BY CUBE(channel_desc, country_id);
```

CHANNEL_DESC	COUNTRY_ID	SALES$
Direct Sales	UK	1,378,126
Direct Sales	US	2,835,557
Direct Sales		**4,213,683**
Internet	UK	911,739
Internet	US	1,732,240
Internet		**2,643,979**
	UK	**2,289,865**
	US	**4,567,797**
		6,857,662

Cube: query and output (example)

User's View

	UK	US	
Direct Sales	1,378,126	2,835,557	**4,213,683**
Internet	911,739	1,732,240	**2,643,979**
	2,289,865	**4,567,797**	**6,857,662**

Rollup: query

- **ROLLUP** gets input in a form of a set of attribute names to be grouped and produces subtotals of rolling-up aggregate combinations of the specified attributes and the grand total.

- The difference with CUBE is that in ROLLUP only *rolling up aggregates are included*, not all possible combinations (refer to example in the next slide).

Rollup: query (example)

```
SELECT channel_desc, calendar_month_desc AS calendar,
       country_id AS co, SUM(amount_sold) AS SALES$
FROM sales, customers, times, channels
WHERE sales.time_id = times.time_id
       AND sales.cust_id = customers.cust_id
       AND sales.channel_id = channels.channel_id
       AND channels.channel_desc IN ('Direct Sales', 'Internet')
       AND times.calendar_month_desc IN ('2000-09', '2000-10')
       AND country_id IN ('UK', 'US')
GROUP BY ROLLUP(channel_desc, calendar_month_desc, country_id);
```

Rollup: output

CHANNEL_DESC	CALENDAR	CO	SALES$
Direct Sales	2000-09	UK	1,378,126
Direct Sales	2000-09	US	2,835,557
Direct Sales	2000-09		**4,213,683**
Direct Sales	2000-10	UK	1,388,051
Direct Sales	2000-10	US	2,908,706
Direct Sales	2000-10		**4,296,757**
Direct Sales			**8,510,440**
Internet	2000-09	UK	911,739
Internet	2000-09	US	1,732,240
Internet	2000-09		**2,643,979**
Internet	2000-10	UK	876,571
Internet	2000-10	US	1,893,753
Internet	2000-10		**2,770,324**
Internet			**5,414,303**
			13,924,743

Cube vs. Rollup

CHANNEL_DESC	CALENDAR	CO	SALES$
Direct Sales	2000-09	UK	1,378,126
Direct Sales	2000-09	US	2,835,557
Direct Sales	2000-09		4,213,683
Direct Sales	2000-10	UK	1,388,051
Direct Sales	2000-10	US	2,908,706
Direct Sales	2000-10		4,296,757
Direct Sales		UK	2,766,177
Direct Sales		US	5,744,263
Direct Sales			8,510,440
Internet	2000-09	UK	911,739
Internet	2000-09	US	1,732,240
Internet	2000-09		2,643,979
Internet	2000-10	UK	876,571
Internet	2000-10	US	1,893,753
Internet	2000-10		2,770,324
Internet		UK	1,788,310
Internet		US	3,625,993
Internet			5,414,303
	2000-09	UK	2,289,865
	2000-09	US	4,567,797
	2000-10	UK	2,264,622
	2000-10	US	4,802,459
		UK	4,554,487
		US	9,370,256
	2000-09		6,857,662
	2000-10		7,067,081
			13,924,743

Cube vs. Rollup

CHANNEL_DESC	CALENDAR	CO	SALES$
Direct Sales	2000-09	UK	1,378,126
Direct Sales	2000-09	US	2,835,557
Direct Sales	2000-09		4,213,683
Direct Sales	2000-10	UK	1,388,051
Direct Sales	2000-10	US	2,908,706
Direct Sales	2000-10		4,296,757
Direct Sales		UK	2,766,177
Direct Sales		US	5,744,263
Direct Sales			8,510,440
Internet	2000-09	UK	911,739
Internet	2000-09	US	1,732,240
Internet	2000-09		2,643,979
Internet	2000-10	UK	876,571
Internet	2000-10	US	1,893,753
Internet	2000-10		2,770,324
Internet		UK	1,788,310
Internet		US	3,625,993
Internet			5,414,303
	2000-09	UK	2,289,865
	2000-09	US	4,567,797
	2000-10	UK	2,264,622
	2000-10	US	4,802,459
		UK	4,554,487
		US	9,370,256
	2000-09		6,857,662
	2000-10		7,067,081
			13,924,743

MONASH University

Cube vs. Rollup (example 2)

```
SELECT channel_desc, country_id, SUM(amount_sold) as SALES$
FROM sales, customers, times, channels
WHERE sales.time_id = times.time_id
      AND sales.cust_id = customers.cust_id
      AND sales.channel_id = channels.channel_id
      AND channels.channel_desc IN ('Direct Sales', 'Internet')
      AND times.calendar_month_desc = '2000-09'
      AND country_id IN ('UK', 'US')
GROUP BY CUBE(channel_desc, country_id);
```

CHANNEL_DESC	COUNTRY_ID	SALES$
Direct Sales	UK	1,378,126
Direct Sales	US	2,835,557
Direct Sales		**4,213,683**
Internet	UK	911,739
Internet	US	1,732,240
Internet		**2,643,979**
	UK	**2,289,865**
	US	**4,567,797**
		6,857,662

MONASH University

Cube vs. Rollup (example 2)

```
SELECT channel_desc, country_id, SUM(amount_sold) as SALES$
FROM sales, customers, times, channels
WHERE sales.time_id = times.time_id
      AND sales.cust_id = customers.cust_id
      AND sales.channel_id = channels.channel_id
      AND channels.channel_desc IN ('Direct Sales', 'Internet')
      AND times.calendar_month_desc = '2000-09'
      AND country_id IN ('UK', 'US')
GROUP BY ROLLUP(channel_desc, country_id);
```

CHANNEL_DESC	COUNTRY_ID	SALES$
Direct Sales	UK	1,378,126
Direct Sales	US	2,835,557
Direct Sales		**4,213,683**
Internet	UK	911,739
Internet	US	1,732,240
Internet		**2,643,979**
	UK	**2,289,865**
	US	**4,567,797**
		6,857,662

MONASH University

Grouping: query

- It is important to differentiate the aggregate NULL values that appear in the output when using CUBE or ROLLUP as opposed to the ‘null’ values when data is not recorded. *In the result of CUBE or ROLLUP, the NULL values represent All aggregate combinations*.

- **GROUPING** clause displays information about which rows are subtotal and for which level of aggregation. It also shows the difference between subtotal values and ‘null’ values.

- GROUPING appears in the SELECT statement list.

Grouping: query (example)

```
SELECT channel_desc, calendar_month_desc as calendar,
       country_id as co, SUM(amount_sold) as SALES$,
       GROUPING(channel_desc) as Ch,
       GROUPING(calendar_month_desc) AS Mo,
       GROUPING(country_id) AS Cou
FROM sales, customers, times, channels
WHERE sales.time_id = times.time_id
       AND sales.cust_id = customers.cust_id
       AND sales.channel_id = channels.channel_id
       AND channels.channel_desc IN ('Direct Sales', 'Internet')
       AND times.calendar_month_desc IN ('2000-09', '2000-10')
       AND country_id IN ('UK', 'US')
GROUP BY ROLLUP(channel_desc, calendar_month_desc, country_id);
```

Grouping: output

CHANNEL_DESC	CALENDAR	CO	SALES$	CH	MO	COU
Direct Sales	2000-09	UK	1,378,126	0	0	0
Direct Sales	2000-09	US	2,835,557	0	0	0
Direct Sales	2000-09		4,213,683	0	0	**1**
Direct Sales	2000-10	UK	1,388,051	0	0	0
Direct Sales	2000-10	US	2,908,706	0	0	0
Direct Sales	2000-10		4,296,757	0	0	**1**
Direct Sales			8,510,440	0	**1**	**1**
Internet	2000-09	UK	911,739	0	0	0
Internet	2000-09	US	1,732,240	0	0	0
Internet	2000-09		2,643,979	0	0	**1**
Internet	2000-10	UK	876,571	0	0	0
Internet	2000-10	US	1,893,753	0	0	0
Internet	2000-10		2,770,324	0	0	**1**
Internet			5,414,303	0	**1**	**1**
			13,924,743	**1**	**1**	**1**

Grouping and Decode: query

DECODE can be used to display appropriate titles for the subtotals

```
SELECT
       DECODE(GROUPING(channel_desc), 1, 'All Channels', channel_desc) AS
Channel,
       DECODE(GROUPING(country_id), 1, 'All Countries', country_id) AS
Country,
       SUM(amount_sold) as SALES$
FROM sales, customers, times, channels
WHERE sales.time_id=times.time_id
       AND sales.cust_id=customers.cust_id
       AND sales.channel_id= channels.channel_id
       AND channels.channel_desc IN ('Direct Sales', 'Internet')
       AND times.calendar_month_desc= '2000-09'
       AND country_id IN ('UK', 'US')
GROUP BY CUBE(channel_desc, country_id);
```

Grouping and Decode: output

CHANNEL	COUNTRY	SALES$
Direct Sales	UK	1,378,126
Direct Sales	US	2,835,557
Direct Sales	**All Countries**	4,213,683
Internet	UK	911,739
Internet	US	1,732,240
Internet	**All Countries**	2,643,979
All Channels	UK	2,289,865
All Channels	US	4,567,797
All Channels	**All Countries**	6,857,662

Grouping and Decode: output

User's View

	UK	US	All Countries
Direct Sales	1,378,126	2,835,557	**4,213,683**
Internet	911,739	1,732,240	**2,643,979**
All Channels	**2,289,865**	**4,567,797**	**6,857,662**

OLAP Queries:

Partial Cube and Partial Rollup

MONASH University

Partial Cube and Partial Rollup

Partial ROLLUP

Rollup to include only some of the subtotals

```
GROUP BY expr1, ROLLUP (expr2, expr3)
```

- First-level subtotals aggregating across **`expr3`** for each combination of **`expr2`** and **`expr1`**
- Second-level subtotals aggregating across **`expr2`** and **`expr3`** for each **`expr1`** value
- No grand total aggregating across all **`expr1, expr2, expr3`**

Basic Rollup

```
SELECT channel_desc, calendar_month_desc AS calendar,
       country_id AS co, SUM(amount_sold) AS SALES$
FROM sales, customers, times, channels
WHERE sales.time_id = times.time_id
       AND sales.cust_id = customers.cust_id
       AND sales.channel_id = channels.channel_id
       AND channels.channel_desc IN ('Direct Sales', 'Internet')
       AND times.calendar_month_desc IN ('2000-09', '2000-10')
       AND country_id IN ('UK', 'US')
GROUP BY ROLLUP(channel_desc, calendar_month_desc, country_id);
```

Basic Rollup

CHANNEL_DESC	CALENDAR	CO	SALES$
Direct Sales	2000-09	UK	1,378,126
Direct Sales	2000-09	US	2,835,557
Direct Sales	2000-09		**4,213,683**
Direct Sales	2000-10	UK	1,388,051
Direct Sales	2000-10	US	2,908,706
Direct Sales	2000-10		**4,296,757**
Direct Sales			**8,510,440**
Internet	2000-09	UK	911,739
Internet	2000-09	US	1,732,240
Internet	2000-09		**2,643,979**
Internet	2000-10	UK	876,571
Internet	2000-10	US	1,893,753
Internet	2000-10		**2,770,324**
Internet			**5,414,303**
			13,924,743

MONASH University

Partial Rollup

```
SELECT channel_desc, calendar_month_desc AS calendar,
       country_id AS co, SUM(amount_sold) AS SALES$
FROM sales, customers, times, channels
WHERE sales.time_id = times.time_id
       AND sales.cust_id = customers.cust_id
       AND sales.channel_id = channels.channel_id
       AND channels.channel_desc IN ('Direct Sales', 'Internet')
       AND times.calendar_month_desc IN ('2000-09', '2000-10')
       AND country_id IN ('UK', 'US')
GROUP BY channel_desc, ROLLUP(calendar_month_desc, country_id);
```

Partial Rollup

CHANNEL_DESC	CALENDAR	CO	SALES$
Direct Sales	2000-09	UK	1,378,126
Direct Sales	2000-09	US	2,835,557
Direct Sales	2000-09		**4,213,683**
Direct Sales	2000-10	UK	1,388,051
Direct Sales	2000-10	US	2,908,706
Direct Sales	2000-10		**4,296,757**
Direct Sales			**8,510,440**
Internet	2000-09	UK	911,739
Internet	2000-09	US	1,732,240
Internet	2000-09		**2,643,979**
Internet	2000-10	UK	876,571
Internet	2000-10	US	1,893,753
Internet	2000-10		**2,770,324**
Internet			**5,414,303**
			~~**13,924,743**~~

Grand total excluded
(empty channel_desc)

MONASH University

Partial Cube and Partial Rollup

Partial CUBE

Partial CUBE resembles partial ROLLUP in that you can limit it to certain dimensions and precede it with columns outside the CUBE operator.

In this case, subtotals of all possible combinations are limited to the dimensions within the cube list (in parentheses), and they are combined with the preceding items in the GROUPBY list.

```
GROUP BY expr1, CUBE (expr2, expr3)
```

Basic Cube

```
SELECT channel_desc, calendar_month_desc AS calendar,
       country_id AS co, SUM(amount_sold) AS SALES$
FROM sales, customers, times, channels
WHERE sales.time_id = times.time_id
       AND sales.cust_id = customers.cust_id
       AND sales.channel_id = channels.channel_id
       AND channels.channel_desc IN ('Direct Sales', 'Internet')
       AND times.calendar_month_desc IN ('2000-09', '2000-10')
       AND country_id IN ('UK', 'US')
GROUP BY CUBE(channel_desc, calendar_month_desc, country_id);
```

Basic Cube

CHANNEL_DESC	CALENDAR	CO	SALES$
Direct Sales	2000-09	UK	1,378,126
Direct Sales	2000-09	US	2,835,557
Direct Sales	2000-09		4,213,683
Direct Sales	2000-10	UK	1,388,051
Direct Sales	2000-10	US	2,908,706
Direct Sales	2000-10		4,296,757
Direct Sales		UK	2,766,177
Direct Sales		US	5,744,263
Direct Sales			8,510,440
Internet	2000-09	UK	911,739
Internet	2000-09	US	1,732,240
Internet	2000-09		2,643,979
Internet	2000-10	UK	876,571
Internet	2000-10	US	1,893,753
Internet	2000-10		2,770,324
Internet		UK	1,788,310
Internet		US	3,625,993
Internet			5,414,303
	2000-09	UK	2,289,865
	2000-09	US	4,567,797
	2000-10	UK	2,264,622
	2000-10	US	4,802,459
		UK	4,554,487
		US	9,370,256
	2000-09		6,857,662
	2000-10		7,067,081
			13,924,743

MONASH University

Partial Cube

```
SELECT channel_desc, calendar_month_desc AS calendar,
       country_id AS co, SUM(amount_sold) AS SALES$
FROM sales, customers, times, channels
WHERE sales.time_id = times.time_id
       AND sales.cust_id = customers.cust_id
       AND sales.channel_id = channels.channel_id
       AND channels.channel_desc IN ('Direct Sales', 'Internet')
       AND times.calendar_month_desc IN ('2000-09', '2000-10')
       AND country_id IN ('UK', 'US')
GROUP BY channel_desc, CUBE(calendar_month_desc, country_id);
```

Partial Cube

CHANNEL_DESC	CALENDAR	CO	SALES$
Direct Sales	2000-09	UK	1,378,126
Direct Sales	2000-09	US	2,835,557
Direct Sales	2000-09		4,213,683
Direct Sales	2000-10	UK	1,388,051
Direct Sales	2000-10	US	2,908,706
Direct Sales	2000-10		4,296,757
Direct Sales		UK	2,766,177
Direct Sales		US	5,744,263
Direct Sales			8,510,440
Internet	2000-09	UK	911,739
Internet	2000-09	US	1,732,240
Internet	2000-09		2,643,979
Internet	2000-10	UK	876,571
Internet	2000-10	US	1,893,753
Internet	2000-10		2,770,324
Internet		UK	1,788,310
Internet		US	3,625,993
Internet			5,414,303
	2000-09	UK	2,289,865
	2000-09	US	4,567,797
	2000-10	UK	2,264,622
	2000-10	US	4,802,459
		UK	4,554,487
		US	9,370,256
	2000-09		6,857,662
	2000-10		7,067,081
			13,924,743

Excluded
(empty Channel_Desc**)**

MONASH University

Partial Cube vs. Partial Rollup

CHANNEL_DESC	CALENDAR	CO	SALES$
Direct Sales	2000-09	UK	1,378,126
Direct Sales	2000-09	US	2,835,557
Direct Sales	2000-09		4,213,683
Direct Sales	2000-10	UK	1,388,051
Direct Sales	2000-10	US	2,908,706
Direct Sales	2000-10		4,296,757
Direct Sales		UK	2,766,177
Direct Sales		US	5,744,263
Direct Sales			8,510,440
Internet	2000-09	UK	911,739
Internet	2000-09	US	1,732,240
Internet	2000-09		2,643,979
Internet	2000-10	UK	876,571
Internet	2000-10	US	1,893,753
Internet	2000-10		2,770,324
Internet		UK	1,788,310
Internet		US	3,625,993
Internet			5,414,303
	2000-09	UK	2,289,865
	2000-09	US	4,567,797
	2000-10	UK	2,264,622
	2000-10	US	4,802,459
		UK	4,554,487
		US	9,370,256
	2000-09		6,857,662
	2000-10		7,067,081
			13,924,743

Only in Partial CUBE

Excluded (empty Channel_Desc)

MONASH University

OLAP Queries:

Advanced Analysis

- Rank (Rank and Dense Rank)
- Row Number
- Percent Rank
- Cumulative Aggregate and Moving Aggregate

MONASH University

OLAP Queries:

Advanced Analysis

- Rank (Rank and Dense Rank)
- Row Number
- Percent Rank
- Cumulative Aggregate and Moving Aggregate

MONASH University

Advanced Analysis (Rank)

RANK

Computes the **rank** of a record compared to other records in the dataset based on the values of a set of measures, for example finding the top three items sold last year.

RANK

Syntax:

```
RANK( ) OVER ([query_partition_clause] order_by_clause)

DENSE_RANK( ) OVER ([query_partition_clause]
order_by_clause)
```

The difference between RANK and DENSE_RANK is that DENSE_RANK leaves no gaps in ranking sequence when there are ties.

RANK (example)

The ORDER BY clause specifies the measures on which ranking is done and defines the order in which rows are sorted in each group.

Once the data is sorted, ranks are given to each row starting from 1.

```
SELECT channel_desc,
  SUM(amount_sold) as SALES$,
  RANK() OVER (ORDER BY SUM(amount_sold) )AS default_rank,
  RANK() OVER (ORDER BY SUM(amount_sold) DESC) AS custom_rank
FROM sales, products, customers, times, channels
WHERE sales.prod_id=products.prod_id
  AND sales.cust_id=customers.cust_id
  AND sales.time_id=times.time_id
  AND sales.channel_id=channels.channel_id
  AND times.calendar_month_desc IN ('2000-09', '2000-10')
  AND country_id='US'
GROUP BY channel_desc;
```

RANK (output)

CHANNEL_DESC	SALES$	DEFAULT_RANK	CUSTOM_RANK
Direct Sales	5,744,263	5	**1**
Internet	3,625,993	4	**2**
Catalog	1,858,386	3	**3**
Partners	1,500,213	2	**4**
Tele Sales	604,656	1	**5**

RANK vs. DENSE_RANK

```
SELECT channel_desc,
  SUM(amount_sold) as SALES$,
  RANK() OVER (ORDER BY SUM(amount_sold) DESC)
  AS custom_rank
FROM …
WHERE …
GROUP BY channel_desc;
```

CHANNEL_DESC	SALES$	CUSTOM_RANK
Direct Sales	5,744,263	1
Internet	3,625,993	2
Catalog	3,625,993	2
Partners	1,500,213	4
Tele Sales	604,656	5

RANK vs. DENSE_RANK

```
SELECT channel_desc,
  SUM(amount_sold) as SALES$,
  DENSE_RANK() OVER (ORDER BY SUM(amount_sold) DESC)
  AS custom_rank
FROM …
WHERE …
GROUP BY channel_desc;
```

CHANNEL_DESC	SALES$	CUSTOM_RANK
Direct Sales	5,744,263	1
Internet	3,625,993	2
Catalog	3,625,993	2
Partners	1,500,213	3
Tele Sales	604,656	4

RANK Per-Group Using PARTITION BY

The previous example shows RANK without partitioning the groups, which is often used when only **one attribute** is selected to display the ranking.

In cases where we need to display rankings of **multiple attributes**, we will need to **partition** the aggregate so that appropriate ranking can be displayed for each of the specified attributes.

When displaying a rank with multiple attributes, we have the option of displaying one rank on one attribute partitioning only, or to display more than one ranks based on a number of partitioning.

RANK Per-Group Using PARTITION BY (example)

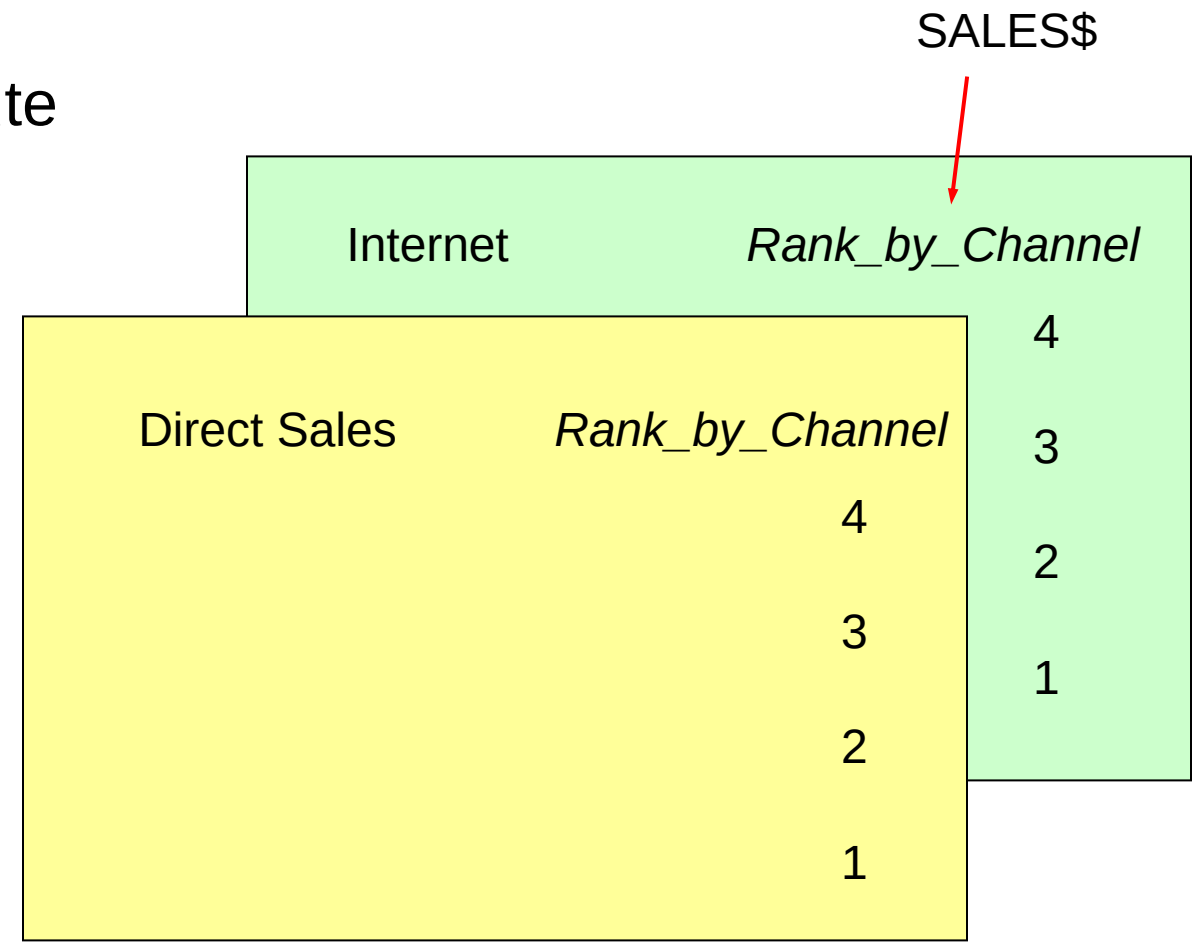
```
SELECT channel_desc, calendar_month_desc AS calendar,
       TO_CHAR(SUM(amount_sold)) AS SALES$,
       RANK() OVER (PARTITION BY channel_desc
       ORDER BY SUM(amount_sold) DESC) AS RANK_BY_CHANNEL
FROM sales, products, customers, times, channels
WHERE sales.prod_id=products.prod_id
       AND sales.cust_id=customers.cust_id
       AND sales.time_id=times.time_id
       AND sales.channel_id=channels.channel_id
       AND times.calendar_month_desc
        IN ('2000-08', '2000-09', '2000-10', '2000-11')
       AND channels.channel_desc
        IN ('Direct Sales', 'Internet')
GROUP BY channel_desc, calendar_month_desc;
```

RANK Per-Group Using PARTITION BY (output)

CHANNEL_DESC	CALENDAR	SALES$	RANK_BY_CHANNEL
Direct Sales	2000-08	9,588,122	**4**
Internet	2000-08	6,084,390	4
Direct Sales	2000-09	9,652,037	**3**
Internet	2000-09	6,147,023	3
Direct Sales	2000-10	10,035,478	**2**
Internet	2000-10	6,417,697	2
Direct Sales	2000-11	12,217,068	**1**
Internet	2000-11	7,821,208	1

RANK Per-Group Using PARTITION BY (visualization)

Partition by one attribute (channel_desc):



RANK Multiple-Groups Using PARTITION BY

The previous example shows PARTITION BY one group only, we can also partition the ranks into more than one group:

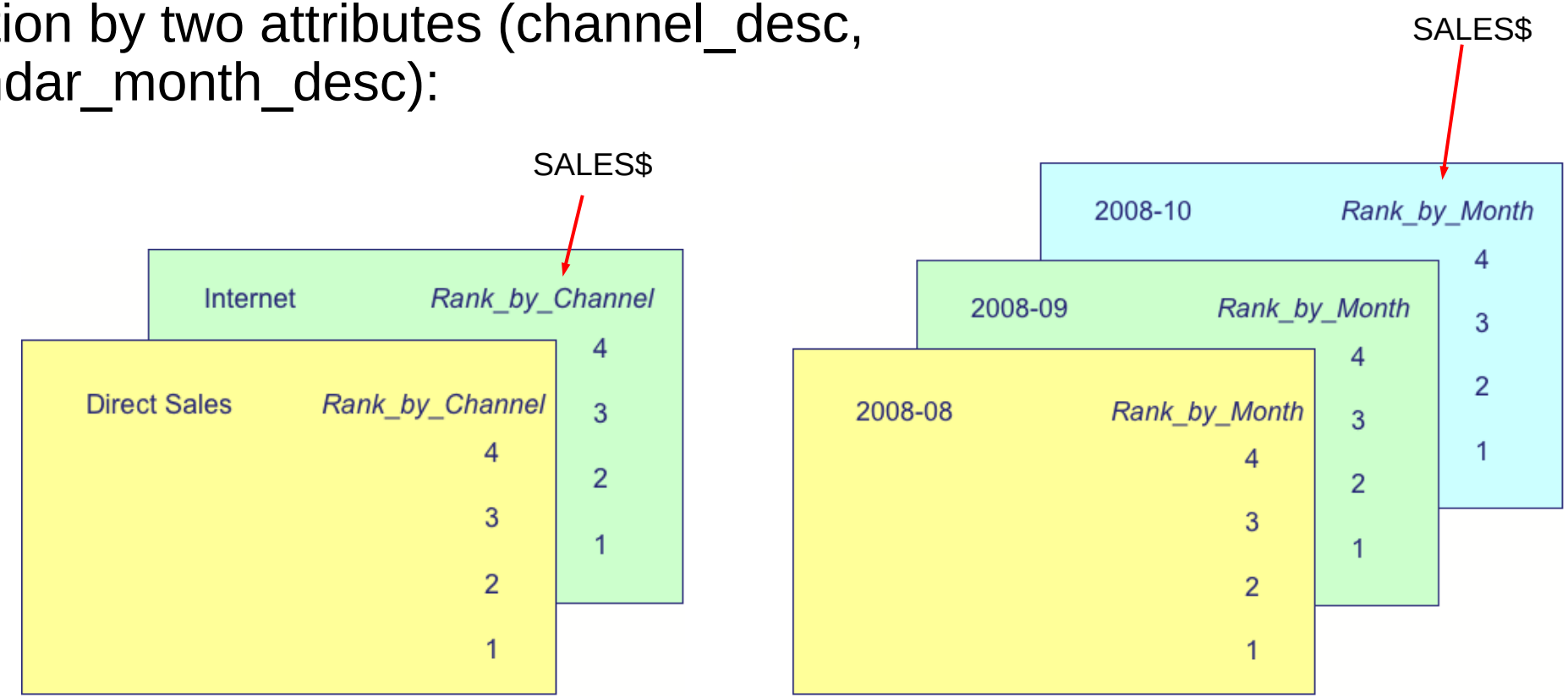
```
SELECT channel_desc, calendar_month_desc AS calendar,
    TO_CHAR(SUM(amount_sold)) AS SALES$,
    RANK() OVER (PARTITION BY channel_desc
    ORDER BY SUM(amount_sold) DESC) AS RANK_BY_CHANNEL,
    RANK() OVER (PARTITION BY calendar_month_desc
    ORDER BY SUM(amount_sold) DESC) AS RANK_BY_MONTH
FROM sales, products, customers, times, channels
WHERE sales.prod_id=products.prod_id
    AND sales.cust_id=customers.cust_id
    AND sales.time_id=times.time_id
    AND sales.channel_id=channels.channel_id
    AND times.calendar_month_desc IN ('2000-08', '2000-09', '2000-10', '2000-
    11')
    AND channels.channel_desc IN ('Direct Sales', 'Internet')
GROUP BY channel_desc, calendar_month_desc;
```

RANK Multiple-Groups Using PARTITION BY

CHANNEL_DESC	CALENDAR	SALES$	RANK_BY_CHANNEL	RANK_BY_MONTH
Direct Sales	2000-08	9,588,122	4	1
Internet	2000-08	6,084,390	4	2
Direct Sales	2000-09	9,652,037	3	1
Internet	2000-09	6,147,023	3	2
Direct Sales	2000-10	10,035,478	2	1
Internet	2000-10	6,417,697	2	2
Direct Sales	2000-11	12,217,068	1	1
Internet	2000-11	7,821,208	1	2

RANK Per-Group Using PARTITION BY (visualization)

Partition by two attributes (channel_desc, calendar_month_desc):



Top N RANKING

Using the RANK function, we can now display **Top N** ranking based on a certain ranking attribute (where N is an integer value, e.g. Top 5).

```
SELECT *
FROM
  (SELECT country_id, SUM(amount_sold) as SALES$,
    RANK() OVER (ORDER BY SUM(amount_sold) DESC ) AS COUNTRY_RANK
   FROM sales,products,customers,times,channels
   WHERE sales.prod_id=products.prod_id
    AND sales.cust_id=customers.cust_id
    AND sales.time_id=times.time_id
    AND sales.channel_id=channels.channel_id
    AND times.calendar_month_desc='2000-09'
    GROUP BY country_id)
WHERE COUNTRY_RANK <= 5;
```

Top N RANKING (output)

CO	SALES$	COUNTRY_RANK
US	6,517,786	1
NL	3,447,121	2
UK	3,207,243	3
DE	3,194,765	4
FR	2,125,572	5

OLAP Queries:

Advanced Analysis

- Rank (Rank and Dense Rank)
- **Row Number**
- Percent Rank
- Cumulative Aggregate and Moving Aggregate

MONASH University

Advanced Analysis (Row Number)

Row Number

The ROW_NUMBER function assigns a unique number (sequentially, starting from 1, as defined by ORDER BY) to each row within the partition.

```
ROW_NUMBER() OVER
([query_partition_clause] order_by_clause)
```

ROW_ NUMBER is a non-deterministic function, so each tied value could have its row number switched. To ensure deterministic results, you must order on a unique key.

Row Number

```
SELECT channel_desc, calendar_month_desc AS calendar,
      TO_CHAR(SUM(amount_sold)) AS SALES$,
      ROW_NUMBER() OVER (ORDER BY SUM(amount_sold) DESC)
      AS ROW_NUMBER
FROM sales, products, customers, times, channels
WHERE sales.prod_id=products.prod_id
      AND sales.cust_id=customers.cust_id
      AND sales.time_id=times.time_id
      AND sales.channel_id=channels.channel_id
      AND times.calendar_month_desc IN ('2000-09', '2000-10')
      GROUP BY channel_desc, calendar_month_desc;
```

Row Number

CHANNEL_DESC	CALENDAR	SALES$	**ROW_NUMBER**
Direct Sales	2000-10	10,000,000	1
Direct Sales	2000-09	9,000,000	2
Internet	2000-10	6,000,000	3
Internet	2000-09	6,000,000	4
Catalog	2000-10	3,000,000	5
Catalog	2000-09	3,000,000	6
Partners	2000-10	2,000,000	7
Partners	2000-09	2,000,000	8
Tele Sales	2000-10	1,000,000	9
Tele Sales	2000-09	1,000,000	10

Row Number vs. Rank vs. Dense Rank

CHANNEL_DESC	CALENDAR	SALES$	RANK
Direct Sales	2000-10	10,000,000	1
Direct Sales	2000-09	9,000,000	2
Internet	2000-10	6,000,000	3
Internet	2000-09	6,000,000	3
Catalog	2000-10	3,000,000	5
Catalog	2000-09	3,000,000	5
Partners	2000-10	2,000,000	7
Partners	2000-09	2,000,000	7
Tele Sales	2000-10	1,000,000	9
Tele Sales	2000-09	1,000,000	9

Row Number vs. Rank vs. Dense Rank

CHANNEL_DESC	CALENDAR	SALES$	**DENSE_RANK**
Direct Sales	2000-10	10,000,000	1
Direct Sales	2000-09	9,000,000	2
Internet	2000-10	6,000,000	3
Internet	2000-09	6,000,000	3
Catalog	2000-10	3,000,000	4
Catalog	2000-09	3,000,000	4
Partners	2000-10	2,000,000	5
Partners	2000-09	2,000,000	5
Tele Sales	2000-10	1,000,000	6
Tele Sales	2000-09	1,000,000	6

OLAP Queries:

Advanced Analysis

- Rank (Rank and Dense Rank)
- Row Number
- Percent Rank
- Cumulative Aggregate and Moving Aggregate

MONASH University

Advanced Analysis (Percent Rank)

RANK and DENSE RANK

Computes the **ranking** of a record, and the rank of the record is an **integer**. The top rank record is rank 1, the second is rank 2, etc. For example, find the top **3** items sold last year (e.g. rank 1, rank 2, and rank 3).

PERCENT RANK

Also computes the **ranking** of a record, but in a **percentage** form. For example, find the top **5%** items sold last year.

Advanced Analysis (Percent Rank)

```
SELECT *
FROM (
  SELECT
    t.time_id as "Time Period",
    sum(f.revenue) AS "Revenue"
    percent_rank() over
      (order by sum(f.revenue) desc) as "Percent Rank"
  FROM dw.TIME t, dw.charter_fact f
  WHERE t.time_id = f.time_id
  GROUP BY t.time_id
) WHERE "Percent Rank" < 0.1;
```

Top 10% revenue

Time P	Revenue	Percent Rank
199503	51144.16	0
199408	49775.51	.024390244
199510	48538.01	.048780488
199409	47647.75	.073170732
199703	45872.32	.097560976

Advanced Analysis (Percent Rank)

```
SELECT *
FROM (
  SELECT
    t.time_id as "Time Period",
    sum(f.revenue) AS "Revenue”
    percent_rank() over
      (order by sum(f.revenue)) as "Percent Rank"
  FROM dw.TIME t, dw.charter_fact f
  WHERE t.time_id = f.time_id
  GROUP BY t.time_id
) WHERE "Percent Rank” >= 0.9;
```

Also top 10% revenue

Time P	Revenue	Percent Rank
199703	45872.32	.902439024
199409	47647.75	.926829268
199510	48538.01	.951219512
199408	49775.51	.975609756
199503	51144.16	1

OLAP Queries:

Advanced Analysis

- Rank (Rank and Dense Rank)
- Row Number
- Percent Rank
- Cumulative Aggregate and Moving Aggregate

MONASH University

Advanced Analysis (Cumulative and Moving Aggregates)

Cumulative Aggregate

Calculate cumulative values within each window partition.

Moving Aggregate

Calculate moving aggregate values within each window partition.

Cumulative Aggregate

```
SELECT c.cust_id, t.calendar_quarter_desc,
   TO_CHAR (SUM(amount_sold), '9,999,999,999') AS Q_SALES,
   TO_CHAR (SUM(SUM(amount_sold)) OVER
       (ORDER BY c.cust_id, t.calendar_quarter_desc
        ROWS UNBOUNDED PRECEDING),
       '9,999,999,999') AS CUM_SALES
FROM sales s, times t, customers c
WHERE s.time_id=t.time_id
AND s.cust_id=c.cust_id
AND t.calendar_year=1999
AND c.cust_id = 6380
GROUP BY c.cust_id, t.calendar_quarter_desc;
```

Cumulative Aggregate

CUST_ID	CALENDA	Q_SALES	CUM_SALES
6380	1999-Q1	60,621	60,621
6380	1999-Q2	68,213	128,834
6380	1999-Q3	75,238	204,072
6380	1999-Q4	57,412	261,484

- The analytic function SUM defines, for each row, a window that starts at the beginning of the partition (UNBOUNDED PRECEDING) and ends, by default, at the current row.
- Nested SUMs are needed in this example since we are performing a SUM over a value that is itself a SUM.
- Nested aggregations are used very often in analytic aggregate functions.

Cumulative Aggregate (with Partition)

```
SELECT c.cust_id, t.calendar_quarter_desc,
   TO_CHAR (SUM(amount_sold), '9,999,999,999') AS Q_SALES,
   TO_CHAR (SUM(SUM(amount_sold)) OVER
       (PARTITION BY c.cust_id
        ORDER BY c.cust_id, t.calendar_quarter_desc
        ROWS UNBOUNDED PRECEDING),
       '9,999,999,999') AS CUM_SALES
FROM sales s, times t, customers c
WHERE s.time_id=t.time_id
AND s.cust_id=c.cust_id
AND t.calendar_year=1999
AND c.cust_id IN (6380, 6510)
GROUP BY c.cust_id, t.calendar_quarter_desc;
```

CUST_ID	CALENDA	Q_SALES	CUM_SALES
6380	**1999-Q1**	**60,621**	**60,621**
6380	**1999-Q2**	**68,213**	**128,834**
6380	**1999-Q3**	**75,238**	**204,072**
6380	**1999-Q4**	**57,412**	**261,484**
6510	1999-Q1	63,030	63,030
6510	1999-Q2	74,622	137,652
6510	1999-Q3	69,966	207,617
6510	1999-Q4	63,366	270,983

Moving Aggregate

This example of a time-based window shows, for one customer, the moving average of sales for the current month and preceding two months:

```
SELECT c.cust_id, t.calendar_quarter_desc,
   TO_CHAR (SUM(amount_sold), '9,999,999,999') AS Q_SALES,
   TO_CHAR (AVG(SUM(amount_sold)) OVER
       (ORDER BY c.cust_id, t.calendar_month_desc
        ROWS 2 PRECEDING),
       '9,999,999,999') AS MOVING_3_MONTH_AVG
FROM sales s, times t, customers c
WHERE s.time_id=t.time_id
AND s.cust_id=c.cust_id
AND t.calendar_year=1999
AND c.cust_id IN (6380)
GROUP BY c.cust_id, t.calendar_month_desc;
```

Moving Aggregate

CUST_ID	CALENDAR	SALES	**MOVING_3_MONTH**
6380	1999-01	19,642	**19,642**
6380	1999-02	19,324	**19,483**
6380	1999-03	21,655	20,207
6380	1999-04	27,091	22,690
6380	1999-05	16,367	21,704
6380	1999-06	24,755	22,738
6380	1999-07	31,332	24,152
6380	1999-08	22,835	26,307
6380	1999-09	21,071	25,079
6380	1999-10	19,279	21,062
6380	1999-11	18,206	19,519
6380	1999-12	19,927	19,137

- Note that the first two rows for the three month moving average calculation in the output data are based on a **smaller interval size** than specified because the window calculation cannot reach past the data retrieved by the query.

Business Intelligence Reporting

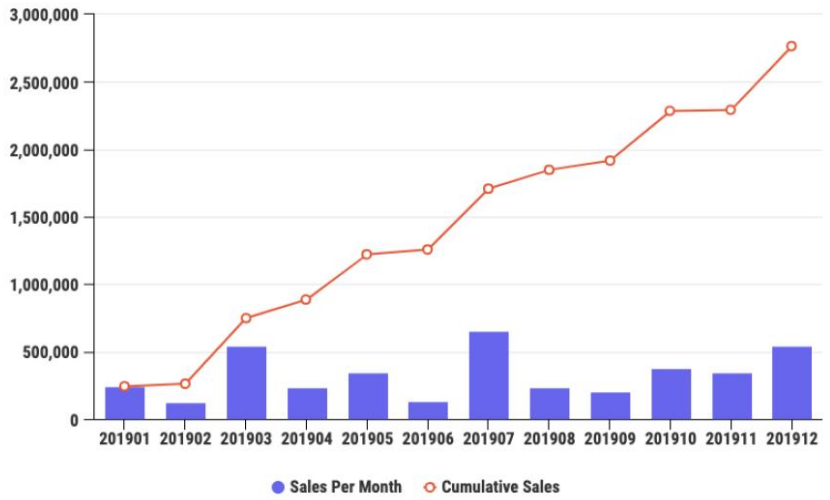
MONASH University

Cumulative and Moving Aggregate

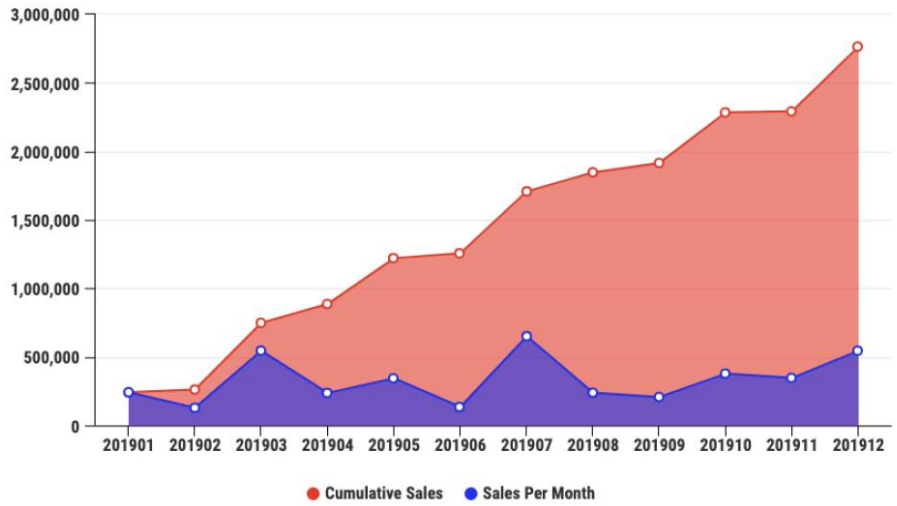
Example:

```
select TimeID, sum(Total_Sales) as Total_Sales,
  sum(sum(Total_Sales)) over (order by TimeID
    rows unbounded preceding) as Cumulative
from SalesFact S, TimeDim T
where S.TimeID = T.TimeID
and Year = 2019
and LocationID in ('MEL')
group by TimeID;
```

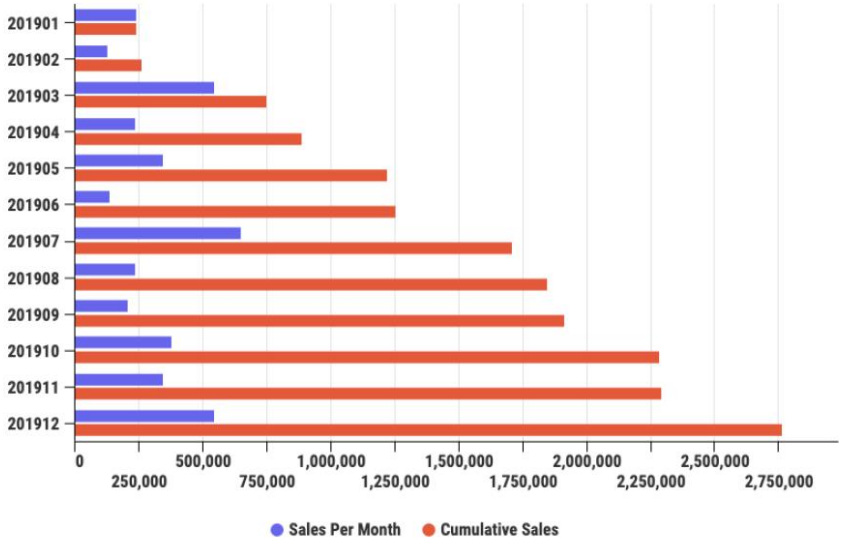
Cumulative and Moving Aggregate



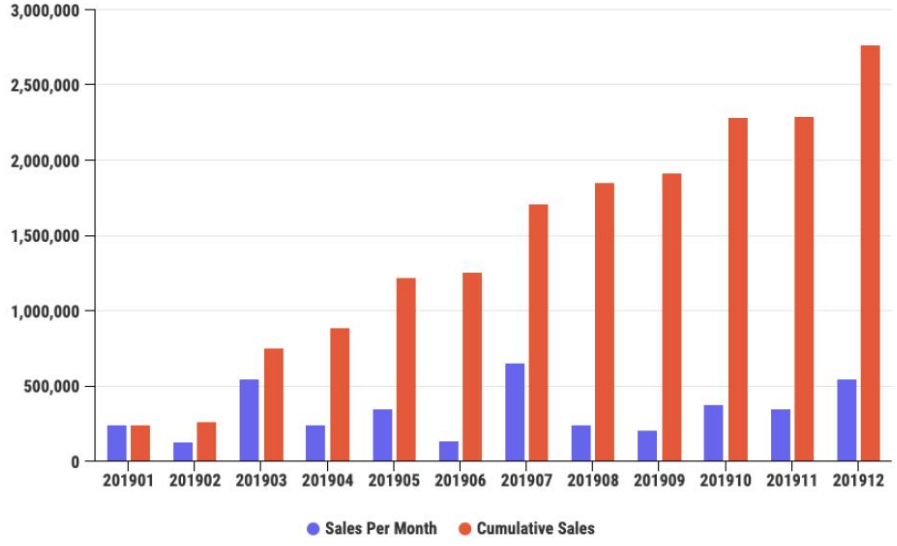
(a)



(b)



(c)



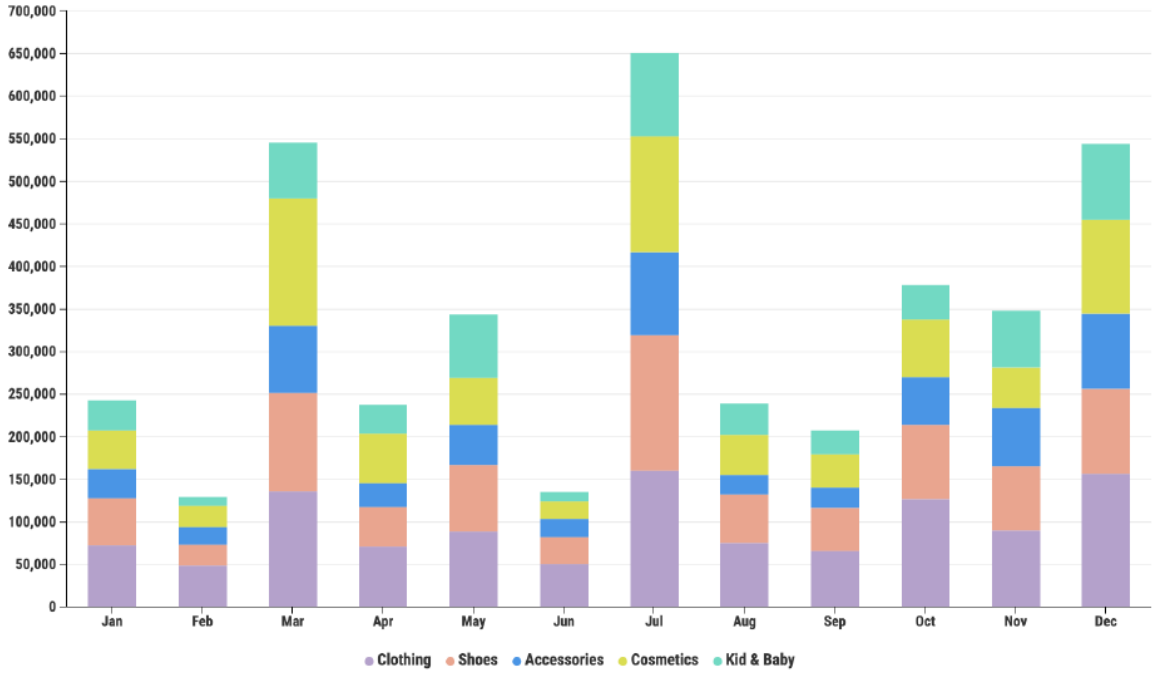
(d)

MONASH University

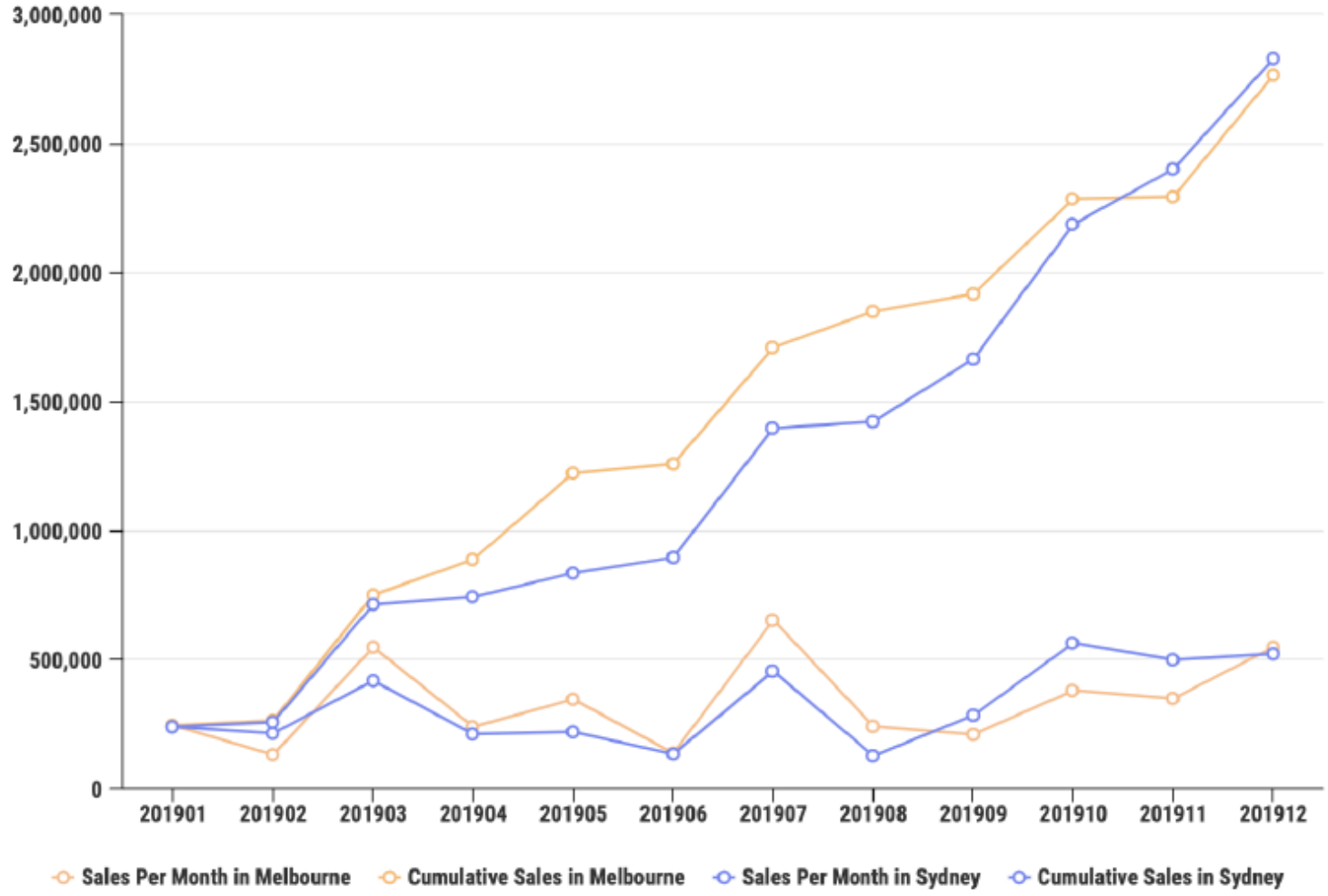
Cumulative and Moving Aggregate

- To drill-down:

```
select S.TimeID, ProductName,
      sum(Total_Sales) as Total_Sales
from SalesFact S, TimeDim T, ProductDim P
where S.TimeID = T.TimeID
and S.ProductID = P.ProductID
and Year = 2019
and LocationID in ('MEL')
group by S.TimeID, ProductName;
```



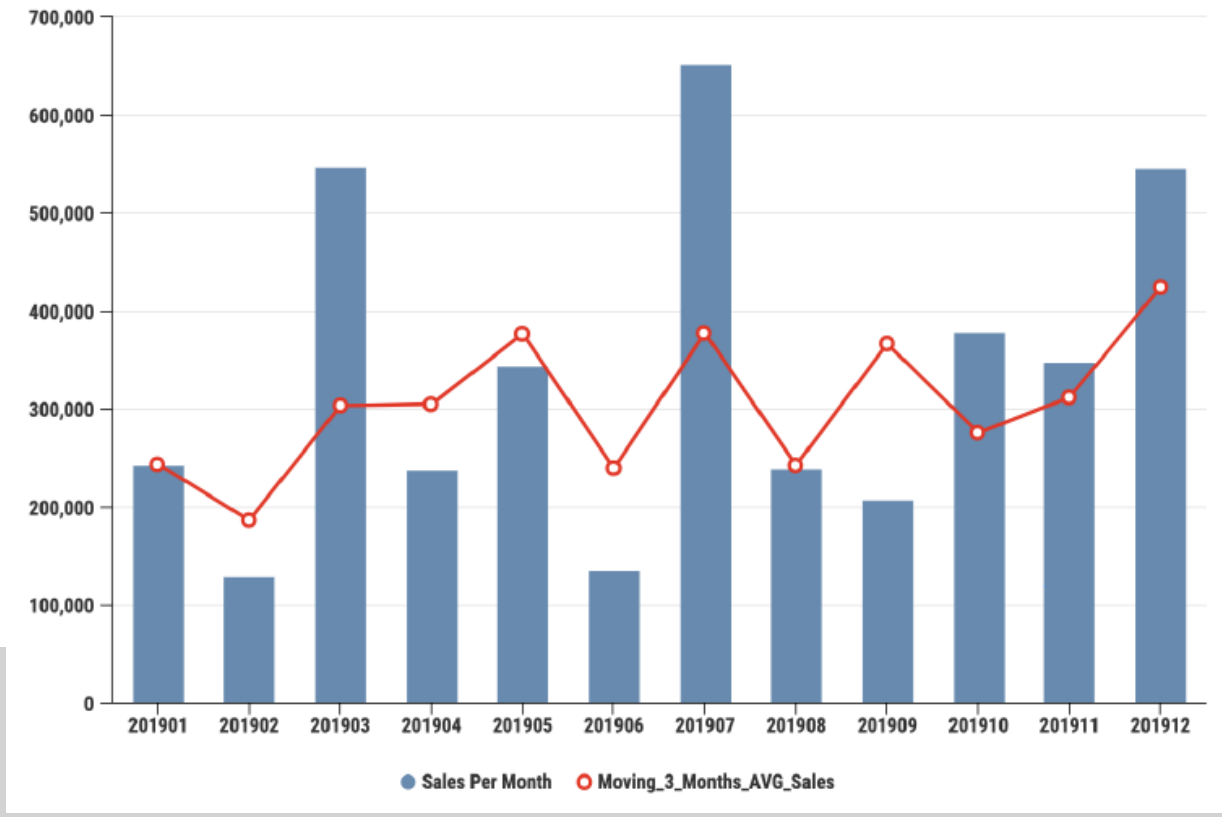
Cumulative and Moving Aggregate



Cumulative and Moving Aggregate

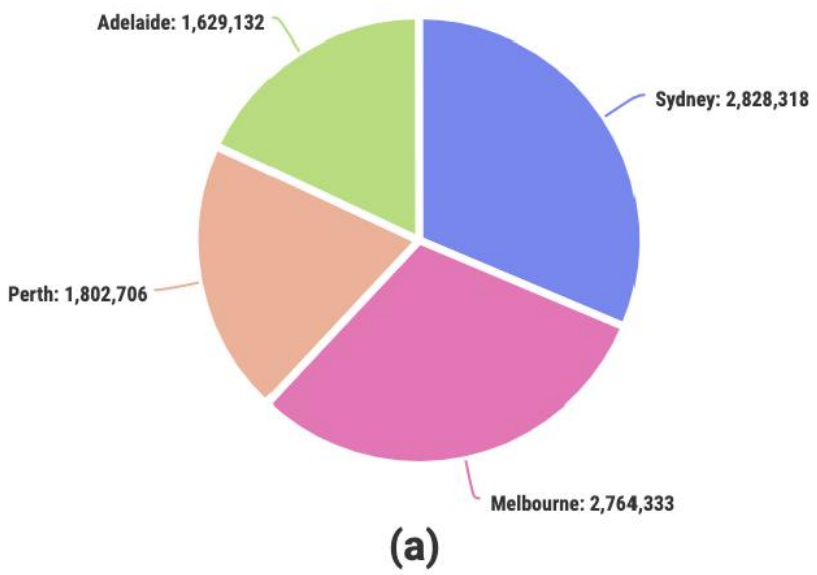
- Moving Aggregate:

```
select S.TimeID, sum(Total_Sales) as Total_Sales,
       avg(sum(Total_Sales)) over (order by S.TimeID rows 2 preceding)
         as Avg_3_Months
from SalesFact S, TimeDim T
where S.TimeID = T.TimeID
and Year = 2019
and LocationID in ('MEL')
group by S.TimeID;
```

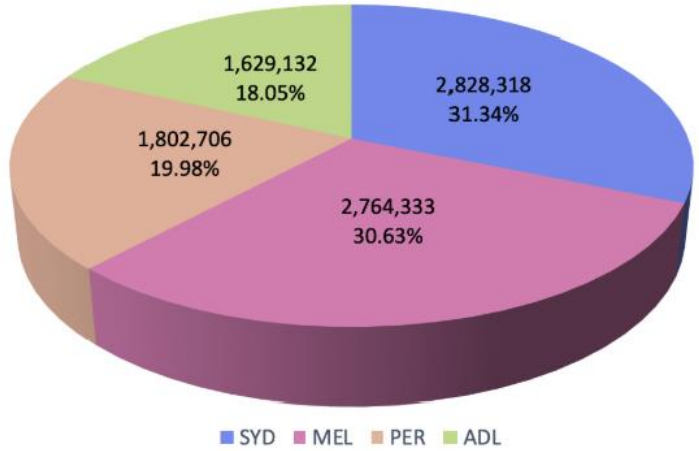


Ratio

```
select Location, sum(Total_Sales) as Total_Sales
from SalesFact S, TimeDim T, LocationDim L
where S.TimeID = T.TimeID
and S.LocationID = L.LocationID
and Year = 2019
group by Location;
```



(a)

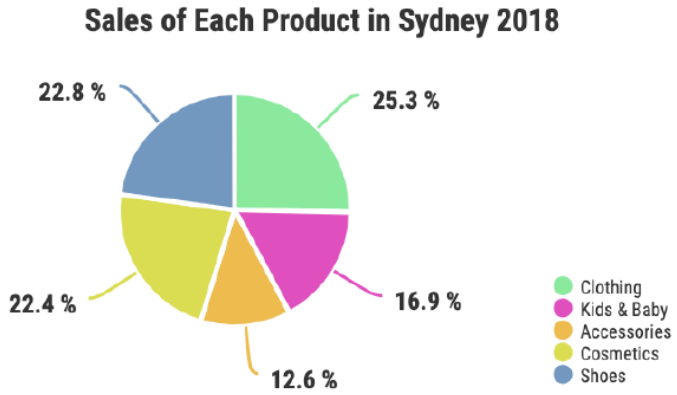


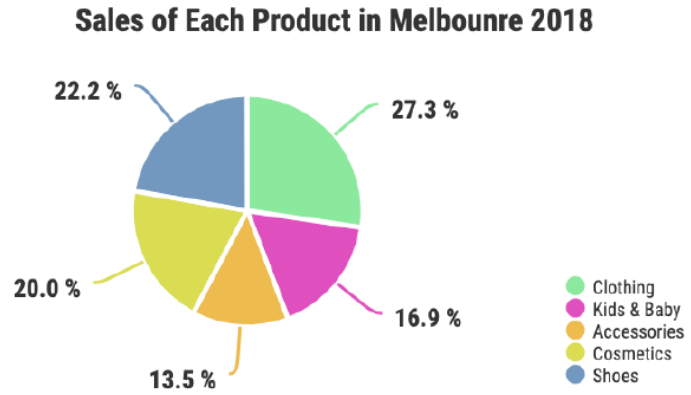
(b)

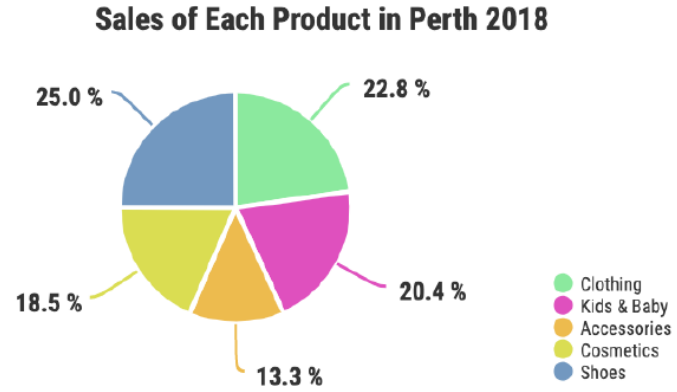
Ratio

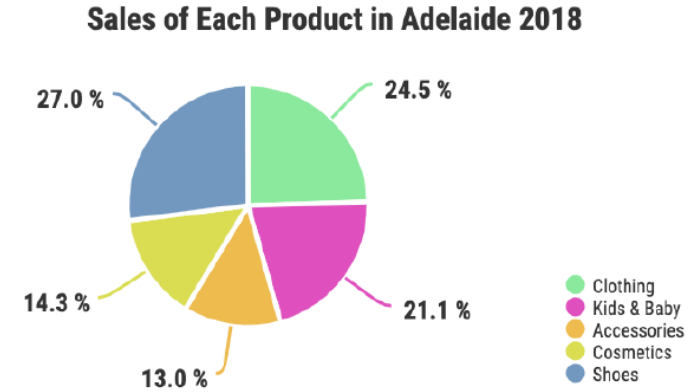
- To drill down into Product level:

```
select Location, ProductName,
     sum(Total_Sales) as Total_Sales
from SalesFact S, TimeDim T,
     LocationDim L, ProductDim P
where S.TimeID = T.TimeID
and S.LocationID = L.LocationID
and S.ProductID = P.ProductID
and Year = 2019
group by Location, ProductName
order by Location, ProductName;
```



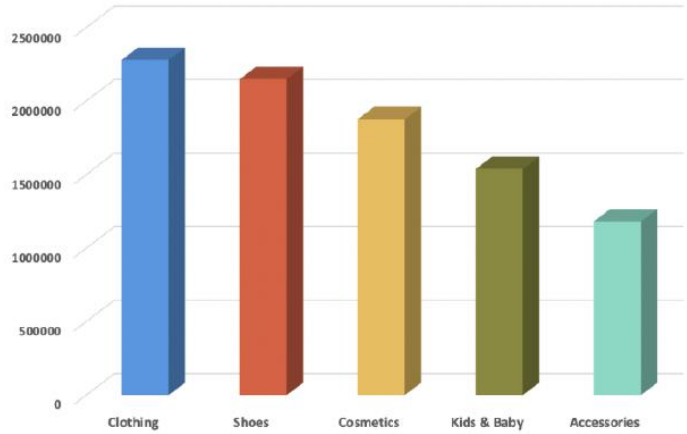




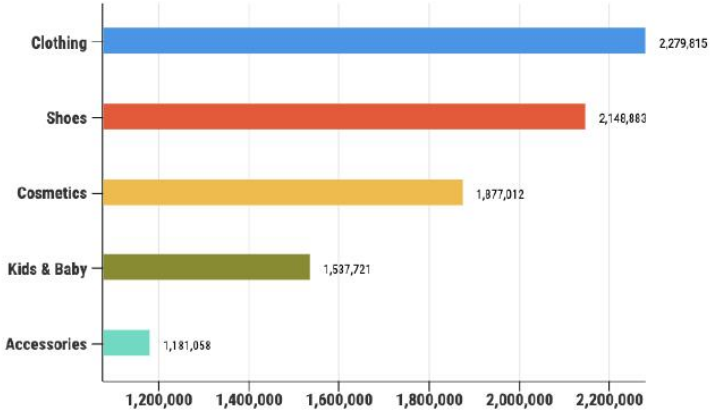


Ranking

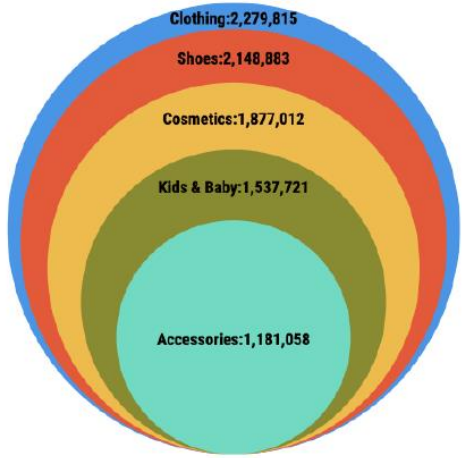
```
select ProductName,
    sum(Total_Sales) as Total_Sales
from SalesFact S,
     TimeDim T,
     ProductDim P
where S.TimeID = T.TimeID
and S.ProductID = P.ProductID
and Year = 2019
group by ProductName
order by ProductName;
```



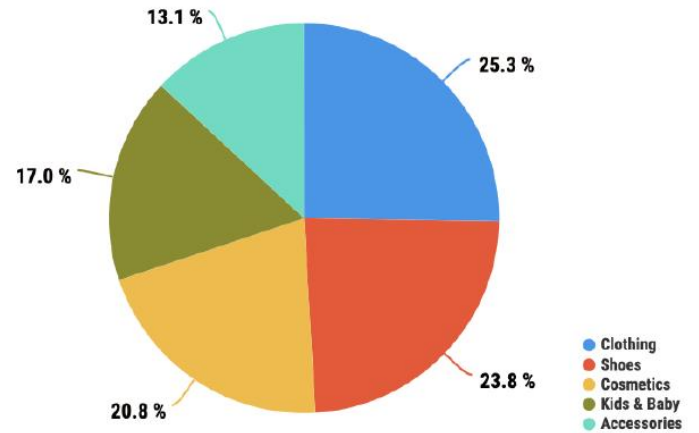
(a)



(b)



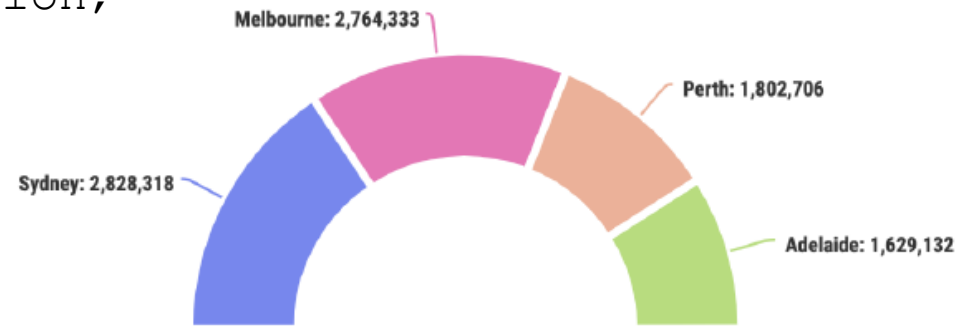
(c)



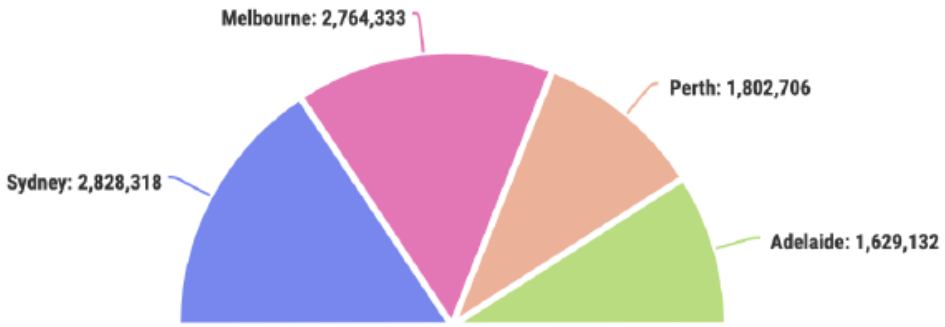
(d)

Ranking

```
select Location, sum(Total_Sales) as Total_Sales
from SalesFact S, TimeDim T, LocationDim L
where S.TimeID = T.TimeID
and S.LocationID = L.LocationID
and Year = 2019
group by Location
order by Location;
```

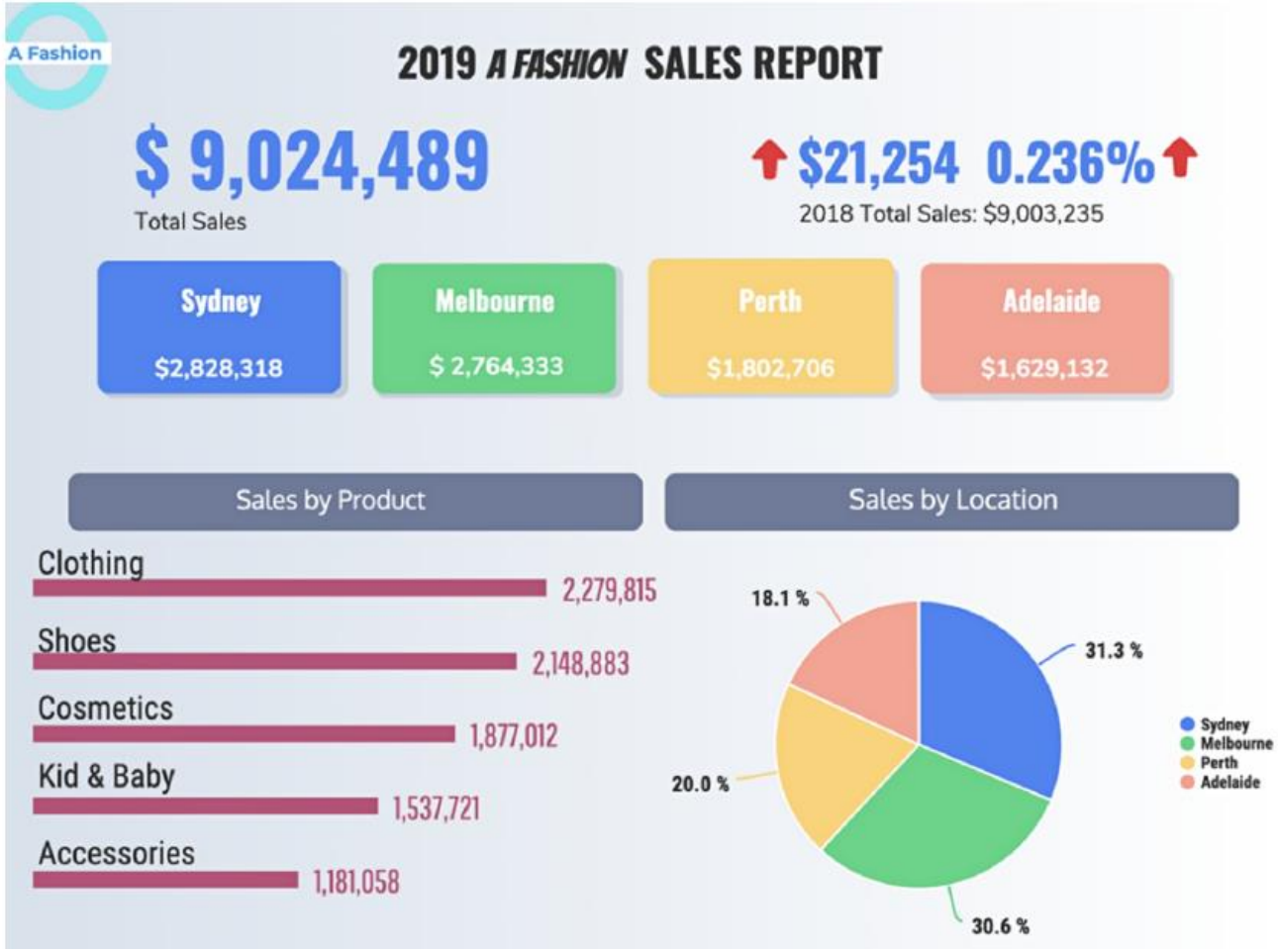


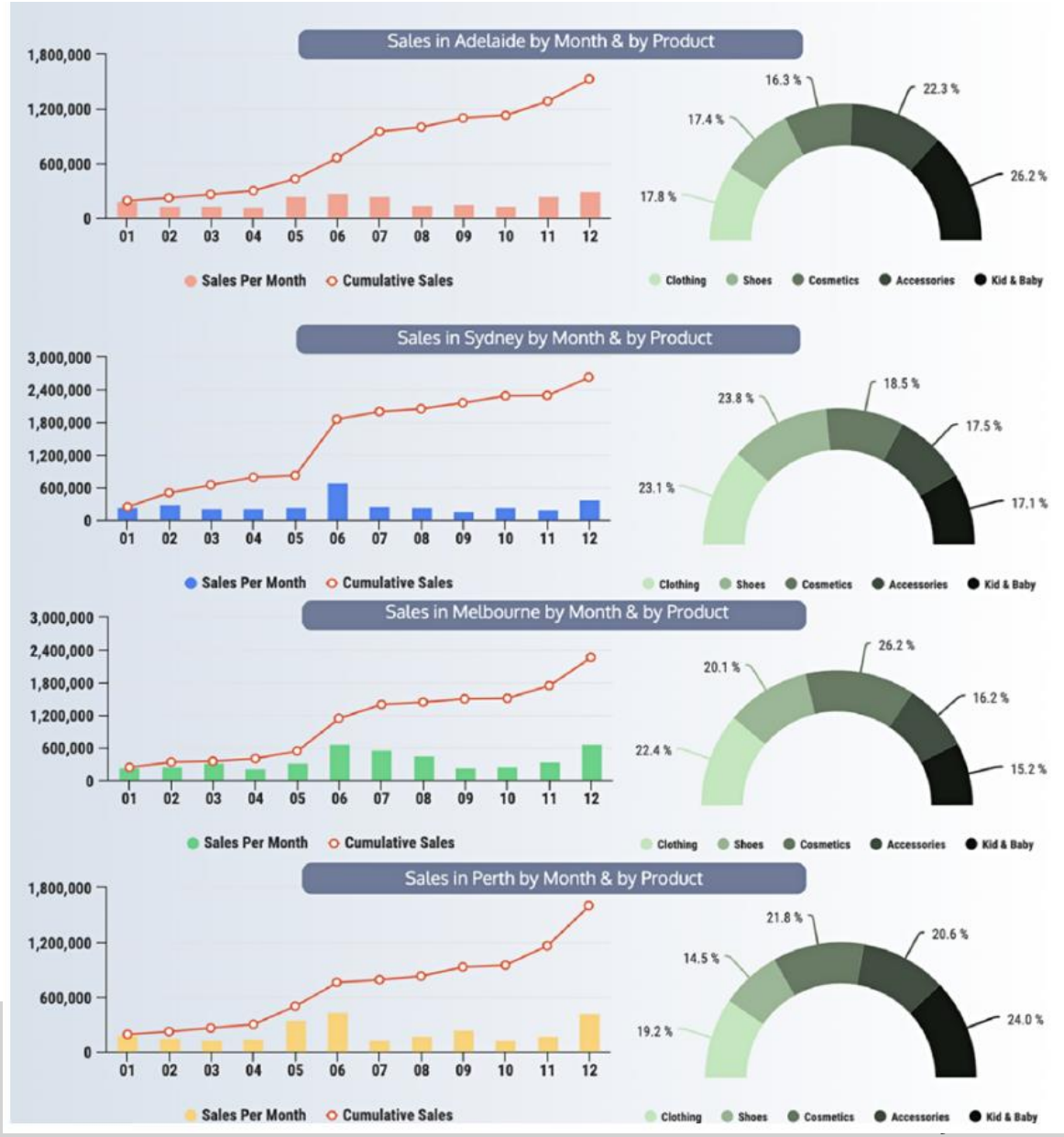
(a)



(b)

A More Complete Report





Summary

MONASH University

Summary – OLAP

- The OLAP queries:
  - **a)** **Basic aggregate functions**: count, sum, avg, max, and min. The group by clause is often used in conjunction with these basic aggregate functions.
  - **b)** **Cube and Rollup**: group by cube, and group by rollup. Simple formatting of the query results can be enhanced thru decode and grouping functions.
  - **c)** **Ranking and Partition**: rank() over and dense_rank() over functions. The row_number() over function has some similarities (as well as differences) to the ranking functions. The partition clause in the ranking function can be used to partition dataset; each with its own ranking.
  - **d)** **Top-N and Top-Percentage Ranking**: use of nested queries to retrieve Top-N, and percent_rank function to retrieve Top-Percentage rankings.
  - **e)** **Cumulative and Moving Aggregate**: row unbounded preceding or row n proceeding can be used to get the cumulative or moving aggregate values.