About this presentation

This presentation is designed to provide an overview of the Kimball Dimensional Modeling methodology and the differences between OLTP and OLAP.

Agenda

- Database crash course
- Dimensional Modeling Overview
- Differences between OLAP and OLTP
- Examples

Fall 2024

What is a database?

A database is an organized collection of structured information, or data, typically stored electronically in a computer system.

-Oracle

Contrary to popular belief, a "database" by this definition can exist in a number of formats:

RDBMS

XML

JSON/NoSQL

Excel

Reference: https://www.oracle.com/database/what-is-database/

Relational Database Terminology

RDBMS: Relational Database Management System

Table: The organizational structure that defines the data elements (columns) and is used to store individual records (rows)

Row (or, tuple): A single record in a database, comprised of data grouped in columns, such as an individual invoice record.

Column: A data element definition that can be used to store a single value per row/record, such as the total amount invoiced.

ER-Model: Entity-Relationship Model, the standard modelling method for relational databases. Leverages the concepts of relationships between database tables.

Relational Database Terminology

Primary Key: A database table column that uniquely identifies an individual record.

Foreign Key: A database column that refers to the primary key of another table.

Let’s visualize this

Table example:

Traffic City	Traffic Is the First Page Viewed Ø (Yes / No)	Traffic Is the Last Page Viewed Ø (Yes / No)	Traffic Page Views	Traffic Site Sessions Ø	Traffic Unique Visitors
Buffalo	No	No	293028	58581	29295
Lancaster	No	No	48111	9383	4244
Hamburg	No	No	32395	5967	2754
North Tonawanda	No	No	24213	4172	1992
Tonawanda	No	No	20453	3578	1696
Lockport	No	No	16514	3012	1371
Niagara Falls	No	No	15060	2937	1312
Orchard Park	No	No	13635	2587	1176
East Aurora	No	No	11620	2128	1000
The Bronx	No	No	11220	2312	1514
Rochester	No	No	9425	1991	1387
Brooklyn	No	No	9218	1947	1292
Grand Island	No	No	8678	1524	750
Alden	No	No	7077	1292	662
Angola	No	No	4373	724	377
Jamestown	No	No	4099	842	505
East Amherst	No	No	3828	661	362
Lake View	No	No	3747	864	503
N/A	No	No	3189	582	286
New York	No	No	3012	637	422
N/A	No	No	2996	558	311
Detroit	No	No	2822	580	352

Let’s visualize this

Column example:

Traffic City	Traffic Is the First Page Viewed Ø (Yes / No)	Traffic Is the Last Page Viewed Ø (Yes / No)	Traffic Page Views	Traffic Site Sessions Ø	Traffic Unique Visitors
Buffalo	No	No	293028	58581	29295
Lancaster	No	No	48111	9383	4244
Hamburg	No	No	32395	5967	2754
North Tonawanda	No	No	24213	4172	1992
Tonawanda	No	No	20453	3578	1696
Lockport	No	No	16514	3012	1371
Niagara Falls	No	No	15060	2937	1312
Orchard Park	No	No	13635	2587	1176
East Aurora	No	No	11620	2128	1000
The Bronx	No	No	11220	2312	1514
Rochester	No	No	9425	1991	1387
Brooklyn	No	No	9218	1947	1292
Grand Island	No	No	8678	1524	750
Alden	No	No	7077	1292	662
Angola	No	No	4373	724	377
Jamestown	No	No	4099	842	505
East Amherst	No	No	3828	661	362
Lake View	No	No	3747	864	503
N/A	No	No	3189	582	286
New York	No	No	3012	637	422
N/A	No	No	2996	558	311
Detroit	No	No	2822	580	352

Let's visualize this

Row example:

Traffic City	Traffic Is the First Page Viewed Ø (Yes / No)	Traffic Is the Last Page Viewed Ø (Yes / No)	Traffic Page Views	Traffic Site Sessions Ø	Traffic Unique Visitors
Buffalo	No	No	293028	58581	29295
Lancaster	No	No	48111	9383	4244
Hamburg	No	No	32395	5967	2754
North Tonawanda	No	No	24213	4172	1992
Tonawanda	No	No	20453	3578	1696
Lockport	No	No	16514	3012	1371
Niagara Falls	No	No	15060	2937	1312
Orchard Park	No	No	13635	2587	1176
East Aurora	No	No	11620	2128	1000
The Bronx	No	No	11220	2312	1514
Rochester	No	No	9425	1991	1387
Brooklyn	No	No	9218	1947	1292
Grand Island	No	No	8678	1524	750
Alden	No	No	7077	1292	662
Angola	No	No	4373	724	377
Jamestown	No	No	4099	842	505
East Amherst	No	No	3828	661	362
Lake View	No	No	3747	864	503
N/A	No	No	3189	582	286
New York	No	No	3012	637	422
N/A	No	No	2996	558	311
Detroit	No	No	2822	580	352

Entity-Relationship Modeling

A logical data model (or, graphical approach) that defines data elements (e.g., tables) and their accompanying relationships. The outcome of this modeling exercise is often referred to as an "ERD" or "Entity Relationship Diagram."

Reference: https://www.guru99.com/er-modeling.html

Let’s visualize this

ER Model:

Student table:

Columns	Data Type
Person#	Integer
Name	Varchar
Advisor#	123

Many

1

Advisor table:

Columns	Data Type
Advisor#	Integer
Name	Varchar
Office Address	Varchar

Let's visualize this

Primary key:

Student table:

Columns	Data Type
Person#	Integer
Name	Varchar
Advisor#	Integer

Many

1

Advisor Table:

Columns	Data Type
Advisor#	Integer
Name	Varchar
Office Address	Varchar

Let’s visualize this

Foreign key:

Student table:

Columns	Data Type
Person#	Integer
Name	Varchar
Advisor#	Integer

Many

1

Advisor Table:

Columns	Data Type
Advisor#	Integer
Name	Varchar
Office Address	Varchar

Let's visualize this

Student table:

Person#	Name	Advisor#
1	Tony Stark	1
2	Betty White	1
3	Jimmy Smith	3

Many

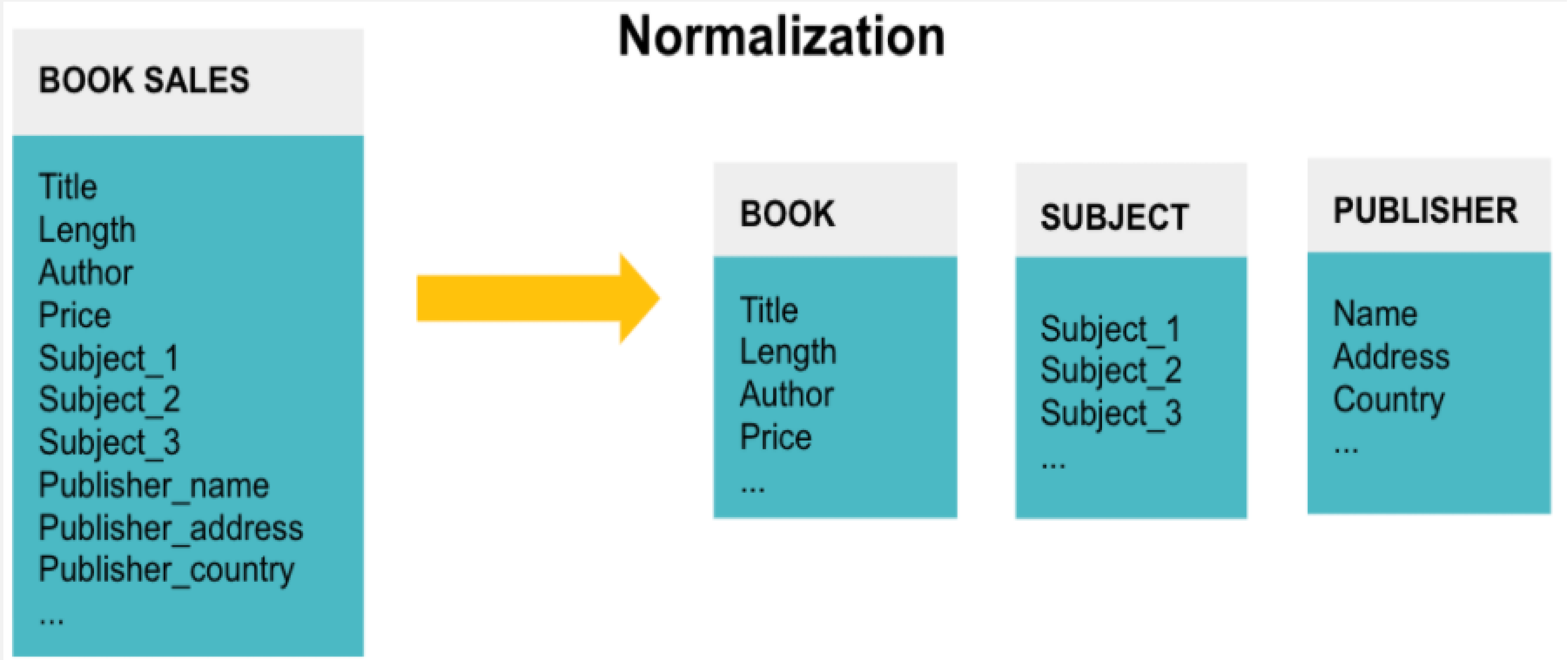
Advisor Table:

1

Advisor#	Name	Office Address
1	John Smith	1234 Student Union
2	Sally Smith	1235 Student Union
3	James Bond	007 Student Union

Normalization

Normalization is, fundamentally, the process of splitting data into many tables (related by one-to-many relationships) to reduce redundancy and consistency errors.



Reference: https://panoply.io/data-warehouse-guide/the-difference-between-a-database-and-a-data-warehouse/

Normalization

Un-normalized

Person#	Advisor	AdvRoom	Class1	Class2	Class3
1022	Jones	412	101-07	143-01	159-02
4123	Smith	216	101-07	143-01	179-04

First Normal Form

Person#	Advisor	AdvRoom	Class#
1022	Jones	412	101-07
1022	Jones	412	143-01
1022	Jones	412	159-02
4123	Smith	216	101-07
4123	Smith	216	143-01
4123	Smith	216	179-04

Reference: https://docs.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description

Normalization

Second Normal Form

Student table:

Person#	Advisor	Adv-Room
1022	Jones	412
4123	Smith	216

Registration table:

Person#	Class#
1022	101-07
1022	143-01
1022	159-02
4123	101-07
4123	143-01
4123	179-04

Reference: https://docs.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description

Normalization

Third Normal Form

Registration table:

Person#	Class#
1022	101-07
1022	143-01
1022	159-02
4123	101-07
4123	143-01
4123	179-04

Student table

Person#	Advisor
1022	Jones
4123	Smith

Faculty table:

Name	Room	Dept
Jones	412	42
Smith	216	42

Reference: https://docs.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description

Normalization

Normalized Database

Student table

Person#	Advisor
1022	Jones
4123	Smith

1

Many

Registration table:

Person#	Class#
1022	101-07
1022	143-01
1022	159-02
4123	101-07
4123	143-01
4123	179-04

Many

Faculty table:

Name	Room	Dept
Jones	412	42
Smith	216	42

1

Reference: https://docs.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description

Querying a database

SQL (Structured Query Language) is the primary method for accessing and manipulating data in a database.

Think of a query as though you are asking a question from the database.

For example:

Question: What are the names of our products and their accompanying prices?

SQL Query: SELECT Names, Price FROM Products_tbl;

Reference: Kimball, Ralph, and Margy Ross. *The Data Warehouse Toolkit : The Definitive Guide to Dimensional Modeling*, John Wiley & Sons, Incorporated, 2013.

Querying a database

Data Definition Language (DDL):

- Used to define and modify the schema and core database elements

Data Control Language (DCL):

- Used to define rights and permissions to users

Data Manipulation Language (DML):

- Used to manipulate data in a database (e.g., add/delete data)

Transaction Control Language (TCL):

- Manages the changes made by DML statements

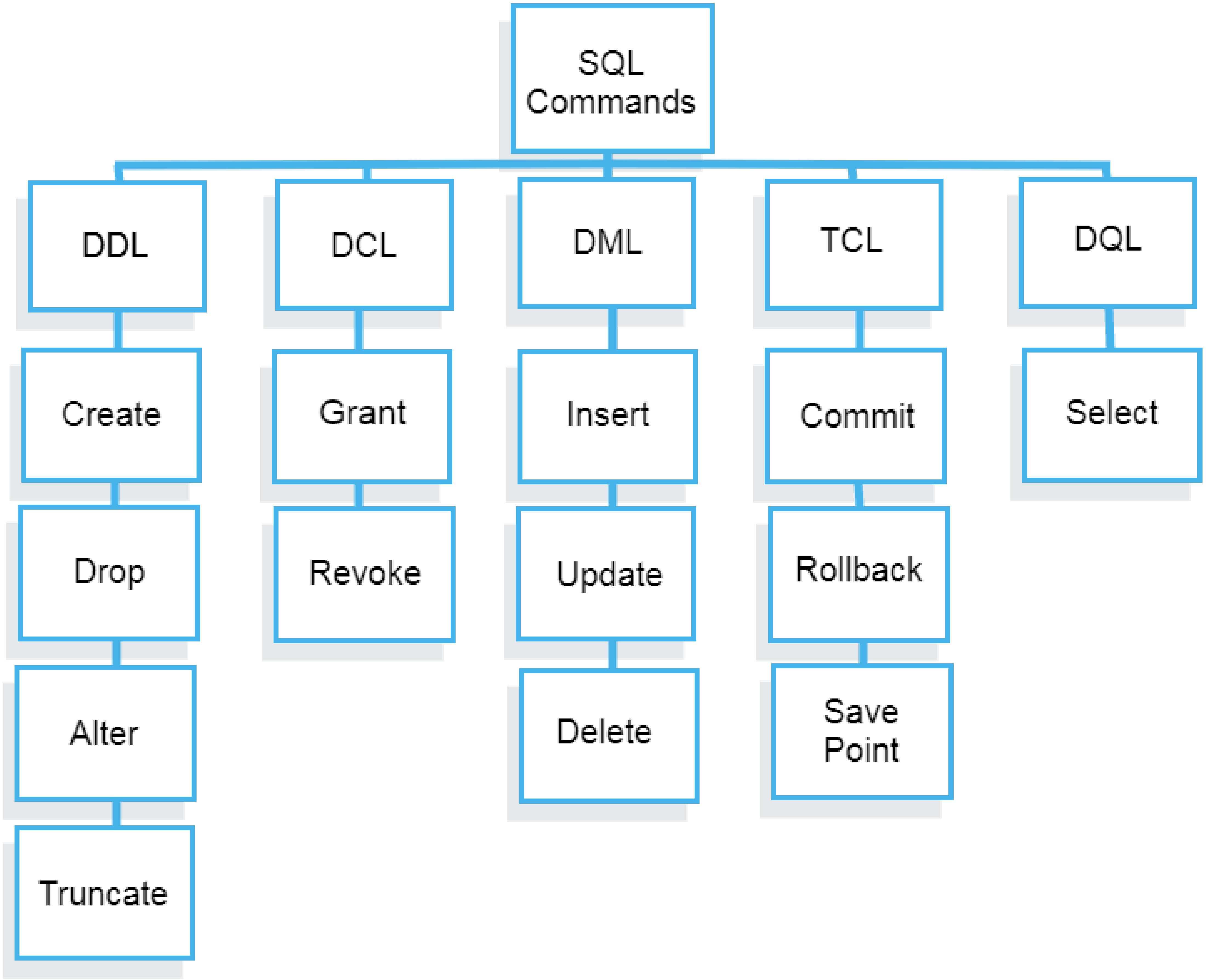
Data Query Language (DQL):

- Used to access (query) data in the database (e.g., select data)

Traditionally, “selecting data” could be considered under DML, but can be separated into DQL, because “selecting” data does not change/manipulate it,

Reference: https://www.guru99.com/sql-commands-dbms-query.html

Querying a database



Reference: https://www.guru99.com/sql-commands-dbms-query.html

Querying a database

Some key operators to know in SQL:

- SELECT: Used to select data from a database, specifying columns after the clause
- INNER JOIN: Combines rows from two or more tables based on a related column (primary and foreign key relationship) where a matching value exists in both tables
- WHERE: Specify a condition to extract records
- * : The “wildcard” operator. Used after a SELECT statement to return all columns in the output.

Reference: w3schools.com/sql

Querying a database

Person#	Name	Advisor#
1	Tony Stark	1
2	Betty White	1
3	Jimmy Smith	3

Many

1

Advisor#	Name	Office_Address
1	John Smith	1234 Student Union
2	Sally Smith	1235 Student Union
3	James Bond	007 Student Union

Question: Who is the advisor of Tony Stark and what is their Office Address?

Answer: John Smith, 1234 Student Union

Querying a database

Person#	Name	Advisor#
1	Tony Stark	1
2	Betty White	1
3	Jimmy Smith	3
4	Sally Sallerson	null

Many

1

Advisor#	Name	Office_Address
1	John Smith	1234 Student Union
2	Sally Smith	1235 Student Union
3	James Bond	007 Student Union

Query: SELECT Person.Name, Advisor.Name, Advisor.Office_Address FROM Person
INNER JOIN Advisor on Person.Advisor# = Advisor.Advisor# WHERE Person.Name='Tony Stark';

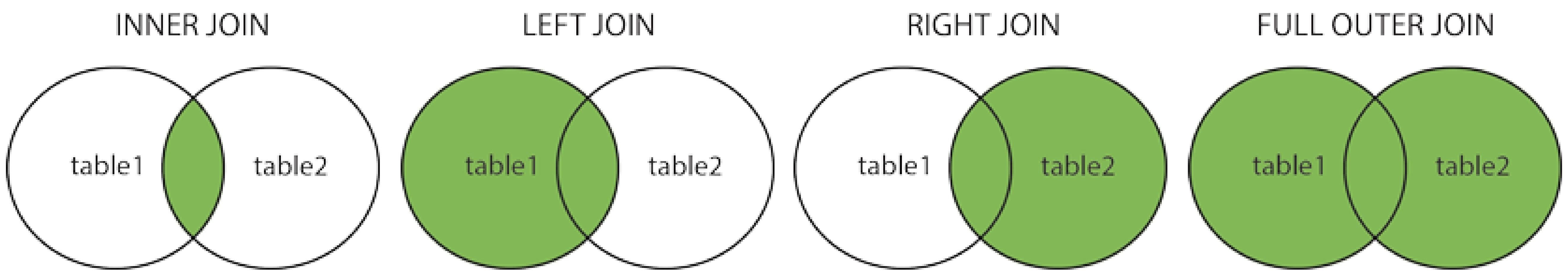
Output:

Person.name	Advisor.name	Advisor.address
Tony Stark	John Smith	1234 Student Union

A bit more on JOINs

Joins are critically important in querying relational databases– we *join* tables together based on their ***relationships***. We can use different joins to return different data.

- INNER JOIN: Combines rows from two or more tables based on a related column (primary and foreign key relationship) where a matching value exists in both tables
- LEFT (outer) JOIN: Returns all records from the "left" table, and any records from the right table that match the left
- RIGHT (outer) JOIN): Returns all records from the "right" table, and any records from the left table that match the right
- FULL (outer) JOIN: Returns all records when there is a match in either table



Reference: w3schools.com/sql

A bit more on JOINs

Why does this matter?

- SELECT Person.Name, Advisor.Name, Advisor.Office_Address FROM Person INNER JOIN Advisor on Person.Advisor# = Advisor.Advisor#;

If we remove the WHERE clause, we will run an INNER JOIN, which will return the following:

Person.name	Advisor.name	Advisor.address
Tony Stark	John Smith	1234 Student Union
Betty White	John Smith	1234 Student Union
Jimmy Smith	James Bond	007 Student Union

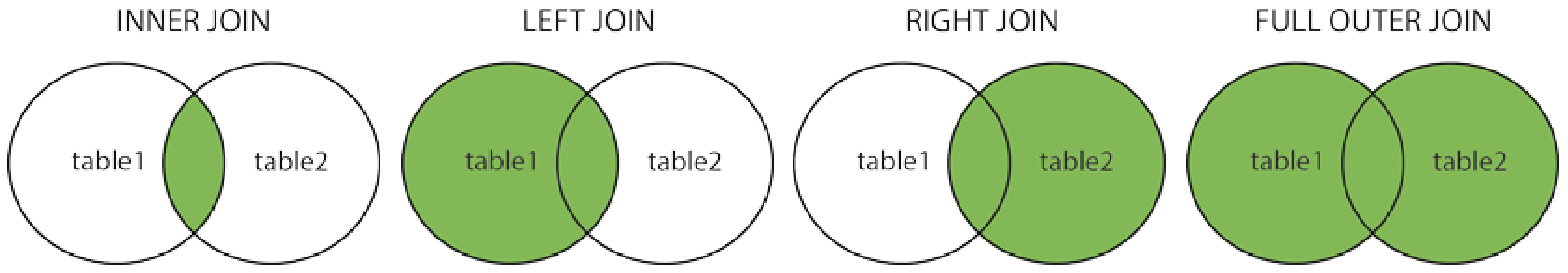
Aren't we missing Sally? ***Why?*** Because an INNER JOIN only returns records where there's a match in ***both*** joined tables. Since Sally doesn't have an advisor, she gets left out!

Person#	Name	Advisor#
1	Tony Stark	1
2	Betty White	1
3	Jimmy Smith	3
4	Sally Sallerson	null

Many

1

Advisor#	Name	Office_Address
1	John Smith	1234 Student Union
2	Sally Smith	1235 Student Union
3	James Bond	007 Student Union



A bit more on JOINs

This is where we get into the use of LEFT, RIGHT, and FULL OUTER JOINs.

- SELECT Person.Name, Advisor.Name, Advisor.Office_Address FROM Person LEFT JOIN Advisor on Person.Advisor# = Advisor.Advisor#;

Our *LEFT* table is the "FROM" table… in this case "Person". So, this will return all records from the person table, even if they don't have a matching value from the "RIGHT" table (Advisor). It will also return matching values from the RIGHT table.

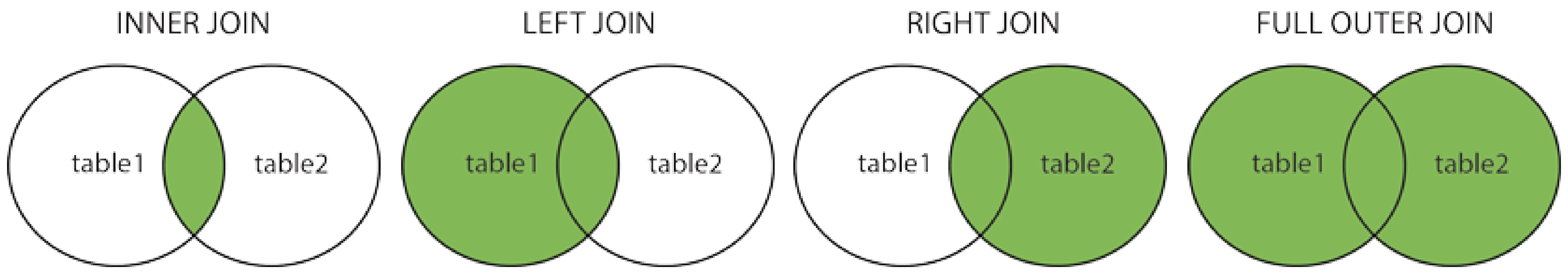
Person.name	Advisor.name	Advisor.address
Tony Stark	John Smith	1234 Student Union
Betty White	John Smith	1234 Student Union
Jimmy Smith	James Bond	007 Student Union
Sally Sallerson	null	null

Person#	Name	Advisor#
1	Tony Stark	1
2	Betty White	1
3	Jimmy Smith	3
4	Sally Sallerson	null

Many

1

Advisor#	Name	Office_Address
1	John Smith	1234 Student Union
2	Sally Smith	1235 Student Union
3	James Bond	007 Student Union



A bit more on JOINs

Inner Join

SQLQuery6.sql - D...-PTJCOP1\Dom (60))* DESKTOP-PTJCOP1.T...ples - dbo.person DESKTOP-PTJCOP1.T...ples - dbo.advisor

```
SELECT Person.Name, Advisor.Name, Advisor.OfficeAddress FROM Person INNER JOIN Advisor on Person.AdvisorNumber = Advisor.AdvisorNumber;
```

100 %

Results Messages

	Name	Name	OfficeAddress
1	Tony Stark	John Smith	1234 Student Union
2	Betty White	John Smith	1234 Student Union
3	Jimmy Smith	James Bond	007 Student Union

Left Join

SQLQuery6.sql - D...-PTJCOP1\Dom (60))* DESKTOP-PTJCOP1.T...ples - dbo.person DESKTOP-PTJCOP1.T...ples - dbo.advisor

```
SELECT Person.Name, Advisor.Name, Advisor.OfficeAddress FROM Person LEFT JOIN Advisor on Person.AdvisorNumber = Advisor.AdvisorNumber;
```

100 %

Results Messages

	Name	Name	OfficeAddress
1	Tony Stark	John Smith	1234 Student Union
2	Betty White	John Smith	1234 Student Union
3	Jimmy Smith	James Bond	007 Student Union
4	Sally Sallerson	NULL	NULL

Right Join

SQLQuery6.sql - D...-PTJCOP1\Dom (60))* DESKTOP-PTJCOP1.T...ples - dbo.person DESKTOP-PTJCOP1.T...ples - dbo.advisor

```
SELECT Person.Name, Advisor.Name, Advisor.OfficeAddress FROM Person RIGHT JOIN Advisor on Person.AdvisorNumber = Advisor.AdvisorNumber;
```

100 %

Results Messages

	Name	Name	OfficeAddress
1	Tony Stark	John Smith	1234 Student Union
2	Betty White	John Smith	1234 Student Union
3	NULL	Sally Smith	1235 Student Union
4	Jimmy Smith	James Bond	007 Student Union

Outer Join

SQLQuery6.sql - D...-PTJCOP1\Dom (60))* DESKTOP-PTJCOP1.T...ples - dbo.person DESKTOP-PTJCOP1.T...ples - dbo.advisor

```
SELECT Person.Name, Advisor.Name, Advisor.OfficeAddress FROM Person FULL OUTER JOIN Advisor on Person.AdvisorNumber = Advisor.AdvisorNumb
```

100 %

Results Messages

	Name	Name	OfficeAddress
1	Tony Stark	John Smith	1234 Student Union
2	Betty White	John Smith	1234 Student Union
3	Jimmy Smith	James Bond	007 Student Union
4	Sally Sallerson	NULL	NULL
5	NULL	Sally Smith	1235 Student Union

Data types

When you build a table in a database, you must select a “data type” for each column. These vary by DBMS platform, but can include:

- int (integer, for a whole number)
- decimal (decimal, for higher-precision non-whole numbers)
- varchar (for variable-character test fields)
- varbinary (for large objects that don’t fit in another category, such as images. Often referred to as a BLOB– binary large object)

Reference: https://docs.microsoft.com/en-us/sql/t-sql/data-types/data-types-transact-sql?view=sql-server-ver15

Dimensional Modeling

Dimensional Modeling is a methodology for designing data warehouses that seeks to deliver data to business users in an understandable way, while providing for fast query performance.

Dimensional modeling is very different from **normalization**, which is a standard database modeling practice that seeks to reduce redundancies.

Reference: Kimball, Ralph, and Margy Ross. *The Data Warehouse Toolkit : The Definitive Guide to Dimensional Modeling*, John Wiley & Sons, Incorporated, 2013.

Key Dimensional Modeling Techniques

The core foundation of a dimensional model is comprised of two primary objects: **facts** and **dimensions**.

Facts (or fact tables) are the measurable outcomes of a business process. For example, a sale or purchase order (including the dollar total) could be considered a fact.

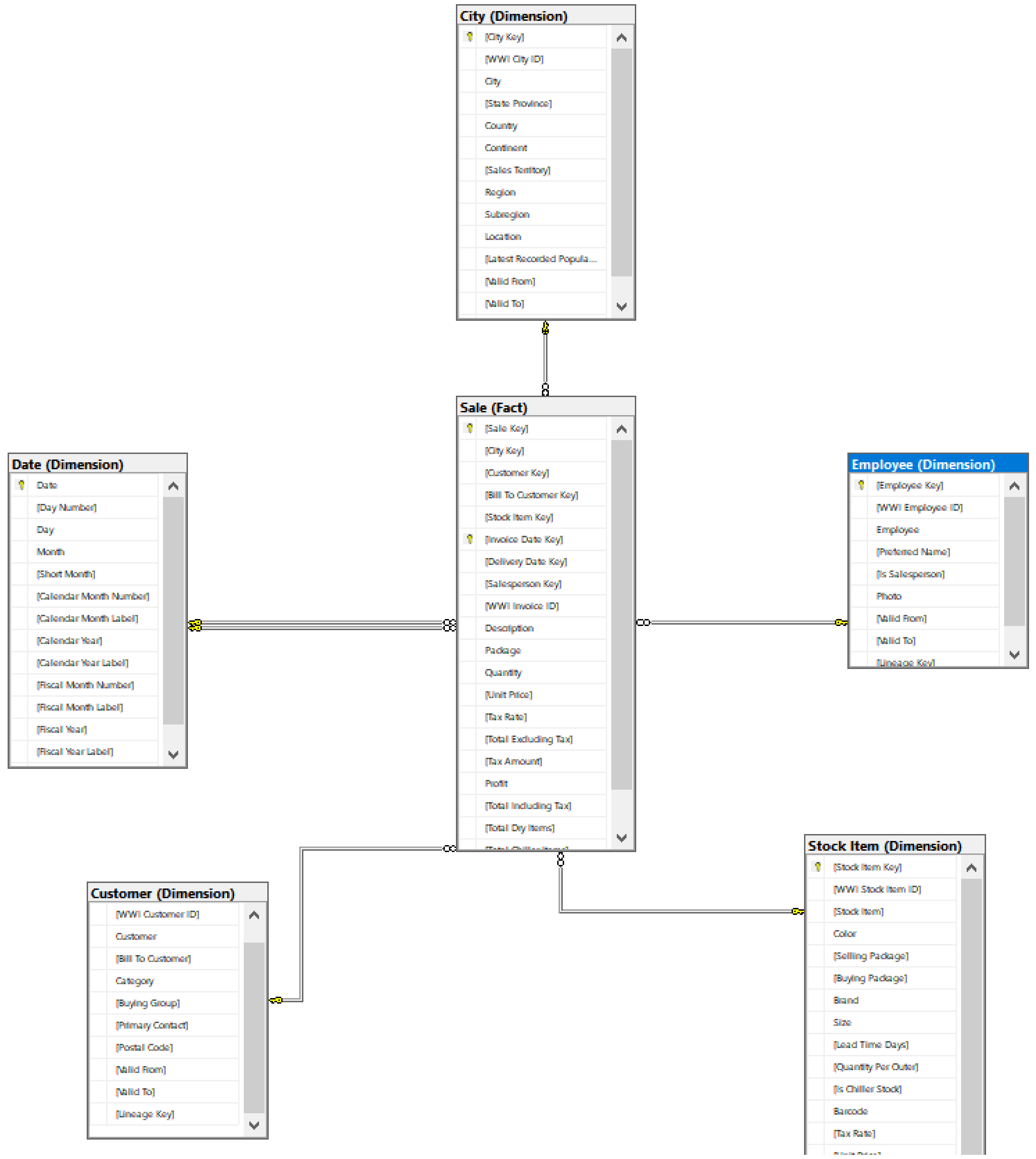
Dimensions (or dimension tables) provide context to facts. For example, the time dimension can provide the time context to a sales fact, while the customer can provide a link to customer information from a sales fact.

Key point: Facts and dimensions are modeled in a 1-to-Many (one dimension row can be associated with many facts, but each fact can only be associated with one row in any given dimension table)

Reference: Kimball, Ralph, and Margy Ross. *The Data Warehouse Toolkit : The Definitive Guide to Dimensional Modeling*, John Wiley & Sons, Incorporated, 2013.

Implementing a Dimensional Model

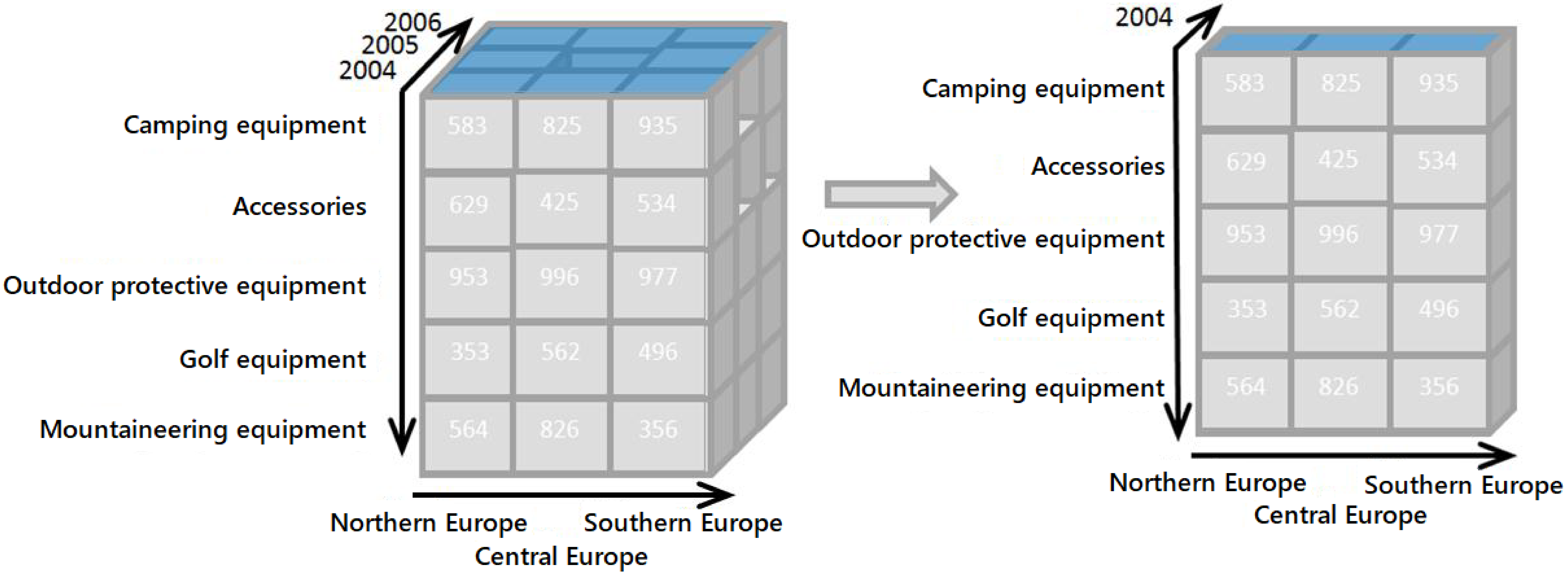
Star Schema: Leverages a relational database platform for implementing the dimensional model.



Reference: Kimball, Ralph, and Margy Ross. *The Data Warehouse Toolkit : The Definitive Guide to Dimensional Modeling*, John Wiley & Sons, Incorporated, 2013.

Implementing a Dimensional Model

OLAP Cube: Leverages a multidimensional database for implementing (or abstracting from) the dimensional model.



Reference: https://en.wikipedia.org/wiki/OLAP_cube#/media/File:OLAP_slicing_en.png

Star Schema VS OLAP Cubes

A few points:

- Star Schema databases are queried with SQL, while OLAP cubes can leverage analytical query languages, like MDX (Multidimensional Expressions)
- OLAP Cubes are often built based off Star Schema relational databases, due to the simplicity of creation of cubes when data has already been dimensionally modeled
- Traditionally, OLAP cubes were favored for analysis performance advantages, but this advantage has lessened as compute performance has improved

The key point: Both Star Schema databases and OLAP Cubes follow the same dimensional modeling practices (Facts and Dimensions).

Reference: Kimball, Ralph, and Margy Ross. *The Data Warehouse Toolkit : The Definitive Guide to Dimensional Modeling*, John Wiley & Sons, Incorporated, 2013.

Star Schema VS OLAP Cubes

"Look, Mom! No JOINs!"

```
SELECT
    { [Measures].[Sales Amount],
        [Measures].[Tax Amount] } ON COLUMNS,
    { [Date].[Fiscal].[Fiscal Year].&[2002],
        [Date].[Fiscal].[Fiscal Year].&[2003] } ON ROWS
FROM [Adventure Works]
WHERE ( [Sales Territory].[Southwest] )
```

MDX query example

Reference: https://docs.microsoft.com/en-us/analysis-services/multidimensional-models/mdx/mdx-query-the-basic-query?view=asallproducts-allversions

Star Schema VS OLAP Cubes

Query goal: *Get sales amount and number of units (on columns) of all product families (on rows) sold in California during Q1 of 2010*

MDX

```
SELECT  {[Measures].[Unit Sales], [Measures].[Store Sales]} ON COLUMNS,
      {[Products].children} ON ROWS
FROM  [Sales]
WHERE ([Time].[2010].[Q1], [Customers].[USA].[CA])
```

SQL

```
SELECT SUM(unit_sales) unit_sales_sum, SUM(store_sales) store_sales_sum
FROM sales
  LEFT JOIN products ON sales.product_id = products.id
  LEFT JOIN product_classes ON products.product_class_id = product_classes.id
  LEFT JOIN time ON sales.time_id = time.id
  LEFT JOIN customers ON sales.customer_id = customers.id
WHERE time.the_year = 2010 AND time.quarter = 'Q1'
  AND customers.country = 'USA' AND customers.state_province = 'CA'
GROUP BY product_classes.product_family
ORDER BY product_classes.product_family
```

Reference: https://dba.stackexchange.com/questions/138311/good-example-of-mdx-vs-sql-for-analytical-queries

"Why don't we just use cubes, then?"

Cubes have limitations!

- They have to be specifically architected by IT to support new business cases, slices of data, drill-down and drill-up.
- They need to be recomputed when underlying data or structure changes– does not support "real-time" use cases very well.

Solution? Use cubes, when needed, as an extension of your dimensional relational database!

“Why don’t we just use a standard normalized database?”

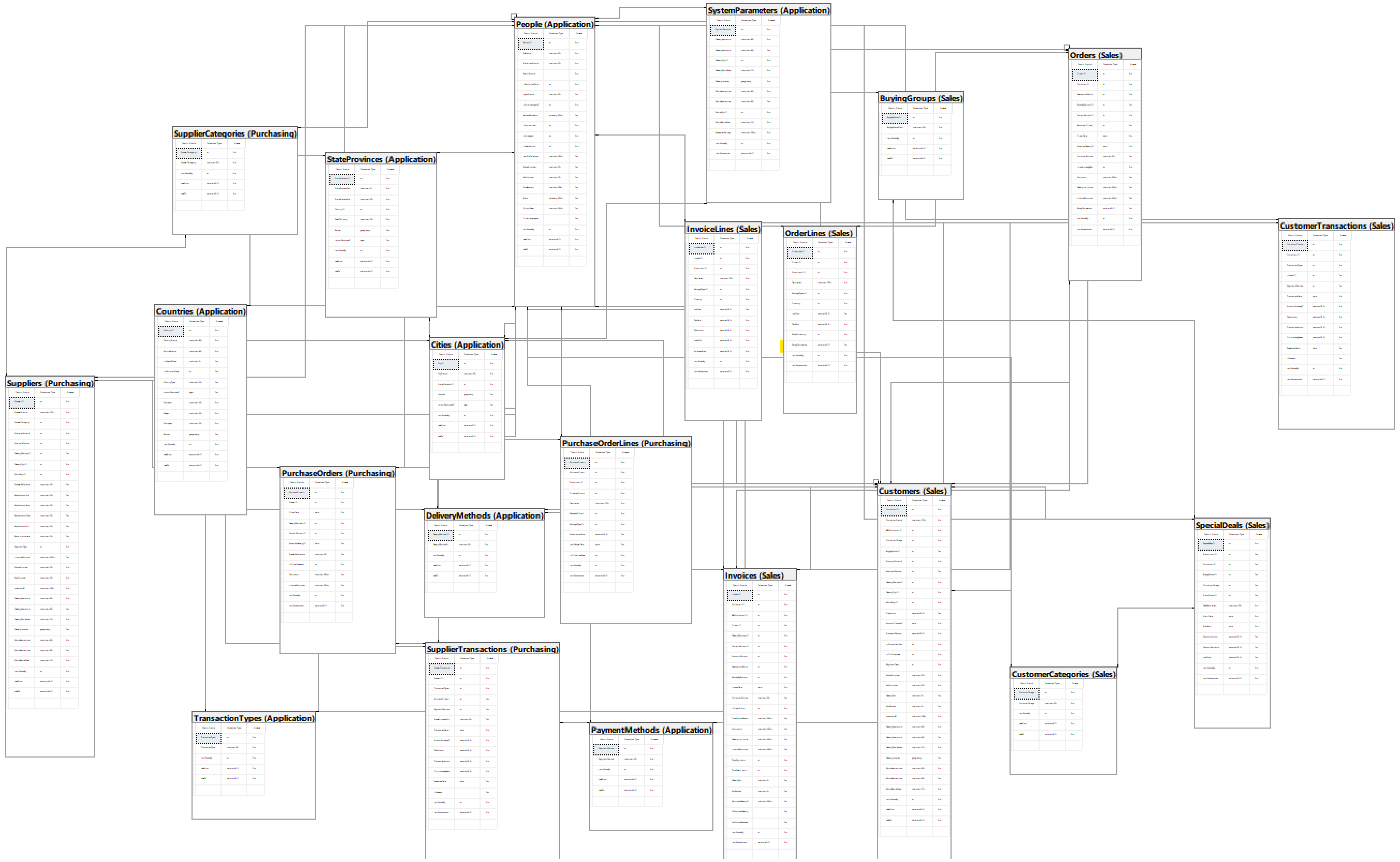
Let’s talk for a bit about normalized databases.

Normalized databases, typically in the third normal form (3NF), are designed to ensure quick, simple operations (e.g., insert, update) while reducing data redundancy and inconsistencies. These databases also largely store “current” information, while archiving old data.

Normalized, transactional databases are the lifeblood of the organization.

“Why don’t we just use a standard normalized database?”

Here is a normalized database for sales tracking:



Reference: Kimball, Ralph, and Margy Ross. *The Data Warehouse Toolkit : The Definitive Guide to Dimensional Modeling*, John Wiley & Sons, Incorporated, 2013.

“Why don’t we just use a standard normalized database?”

voiceLines (Sales)

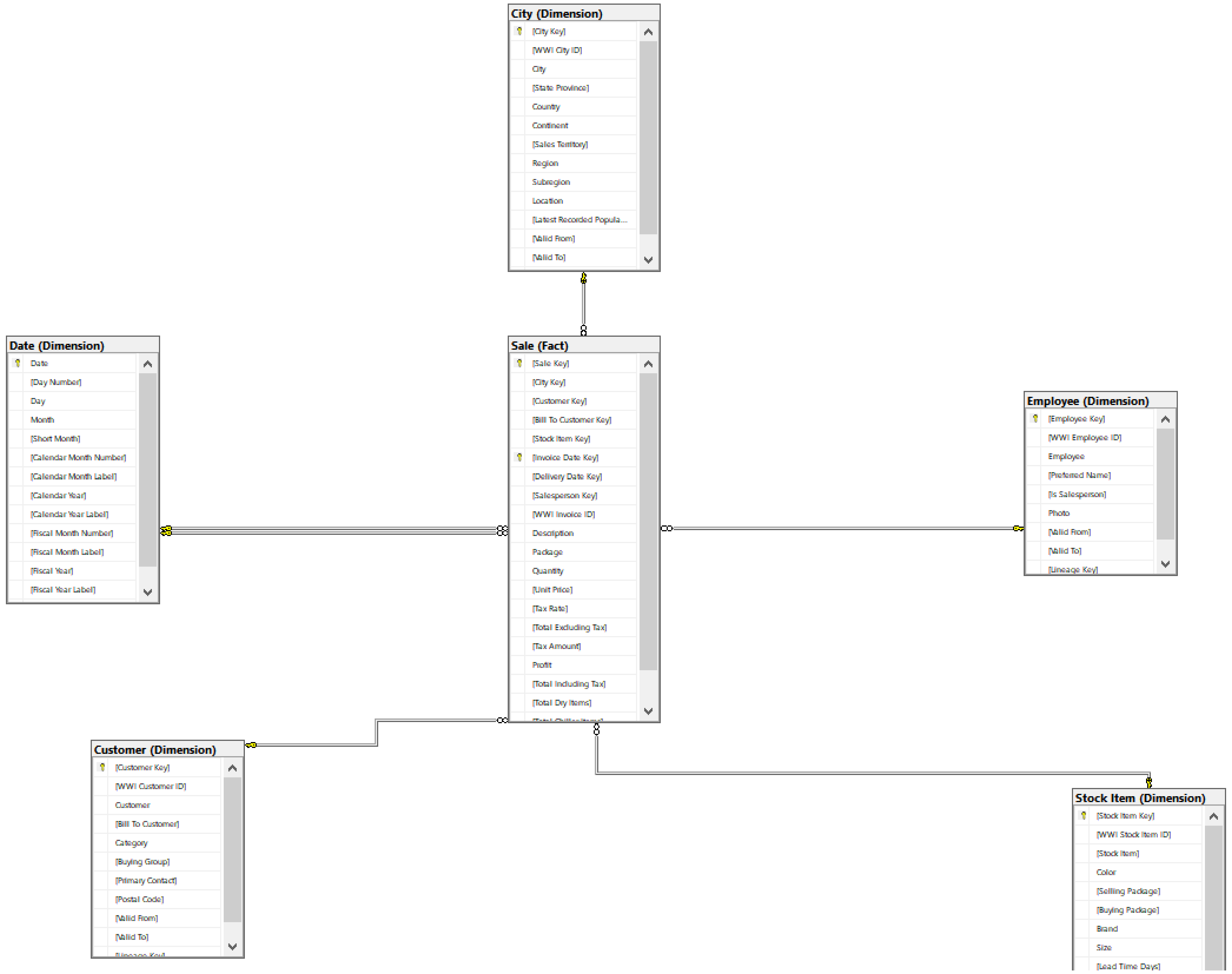
Column Name	Condensed Type	Nullable
InvoiceLineID	int	No
InvoiceID	int	No
StockItemID	int	No
Description	nvarchar(100)	No
PackageTypeID	int	No
Quantity	int	No
UnitPrice	decimal(18, 2)	Yes
TaxRate	decimal(18, 3)	No
TaxAmount	decimal(18, 2)	No
LineProfit	decimal(18, 2)	No
ExtendedPrice	decimal(18, 2)	No
LastEditedBy	int	No
LastEditedWhen	datetime2(7)	No

OrderLines (Sales)

Column Name	Condensed Type	Nullable
OrderLineID	int	No
OrderID	int	No
StockItemID	int	No
Description	nvarchar(100)	No
PackageTypeID	int	No
Quantity	int	No
UnitPrice	decimal(18, 2)	Yes
TaxRate	decimal(18, 3)	No
PickedQuantity	int	No
PickingCompletedWhen	datetime2(7)	Yes
LastEditedBy	int	No
LastEditedWhen	datetime2(7)	No

“Why don’t we just use a standard normalized database?”

Here is a denormalized sales database in star schema:



“Why don’t we just use a standard normalized database?”

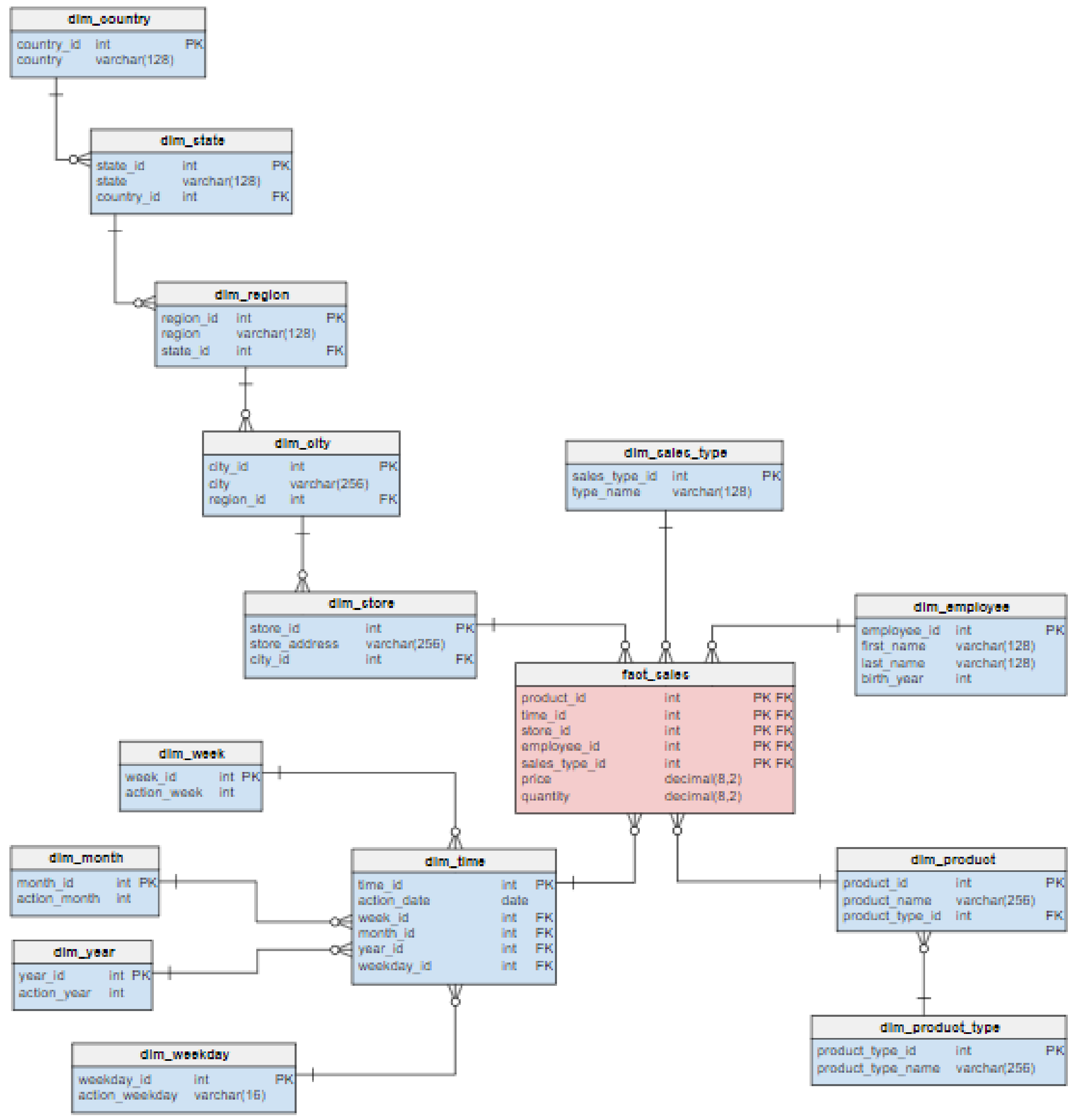
Remember our success criteria!

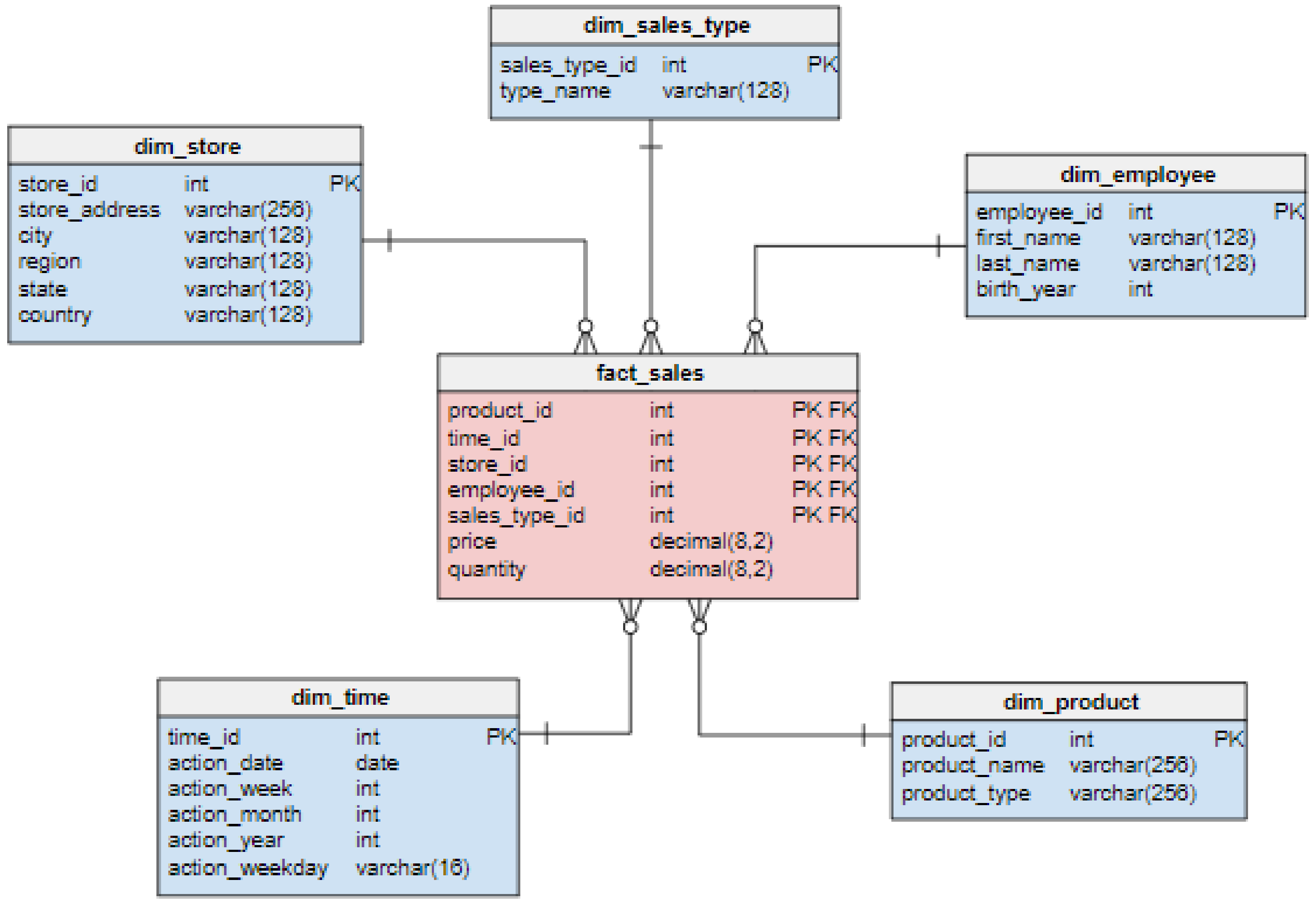
Which database structure:

1. Is more intuitive and understandable for a non-technical user?
2. Is easier to update in the face of changing business reporting requirements?

"Why don't we just use a standard normalized database?"

If you're still not convinced, let's look at the steps to query a *slightly* normalized database (using what's called the Snowflake schema) vs an un-normalized Star Schema database.





Reference: https://vertabelo.com/blog/data-warehouse-modeling-star-schema-vs-snowflake-schema/

“Why don’t we just use a standard normalized database?”

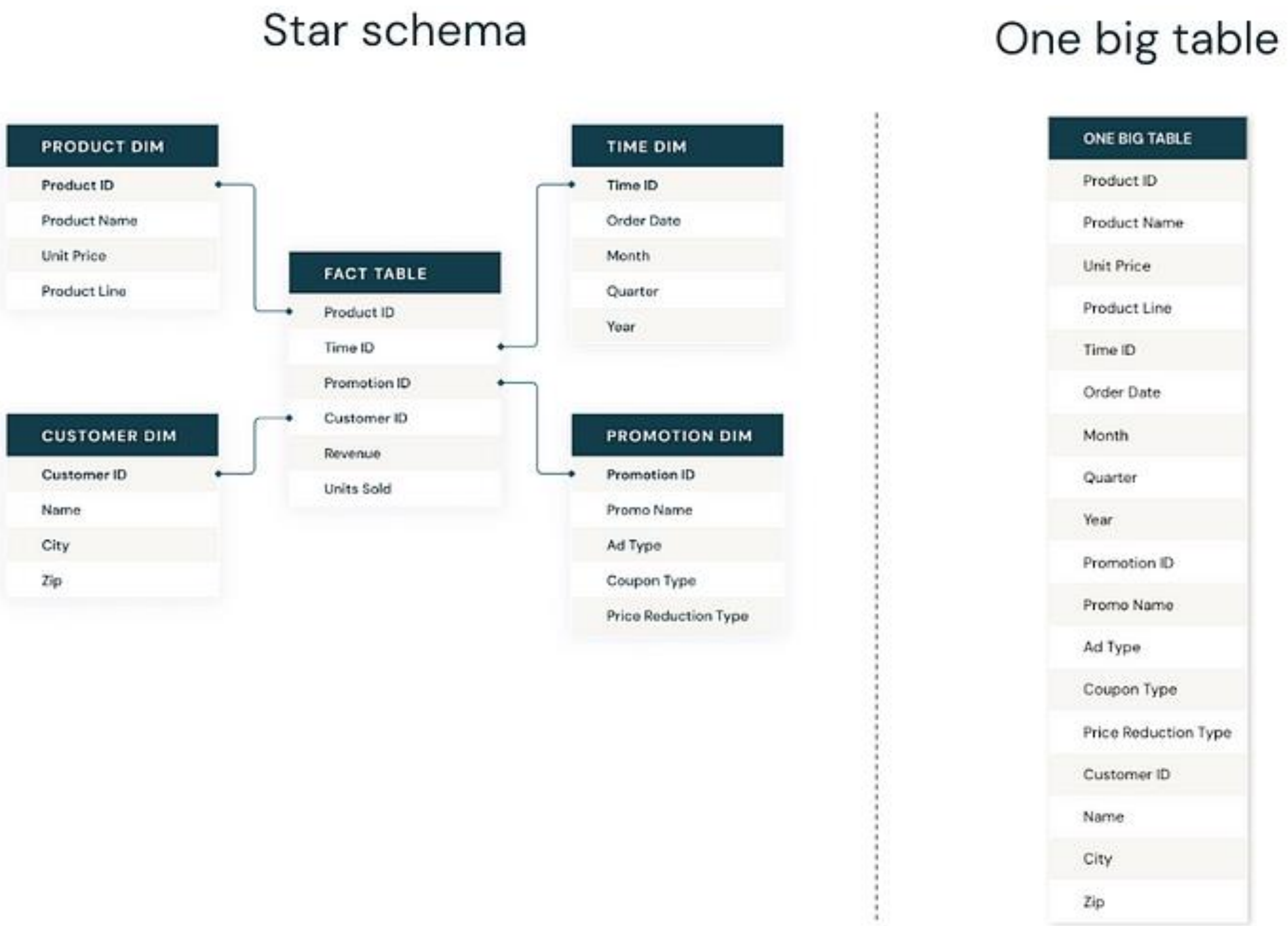
And here’s what the queries to get the same data from each look like…

```
SELECT
  dim_store.store_address,
  SUM(fact_sales.quantity) AS quantity_sold

FROM
  fact_sales
  INNER JOIN dim_product ON fact_sales.product_id = dim_product.product_id
  INNER JOIN dim_product_type ON dim_product.product_type_id = dim_product_type.product_type_id
  INNER JOIN dim_time ON fact_sales.time_id = dim_time.time_id
  INNER JOIN dim_year ON dim_time.year_id = dim_year.year_id
  INNER JOIN dim_store ON fact_sales.store_id = dim_store.store_id
  INNER JOIN dim_city ON dim_store.city_id = dim_city.city_id

WHERE
  dim_year.action_year = 2016
  AND dim_city.city = 'Berlin'
  AND dim_product_type.product_type_name = 'phone'

GROUP BY
  dim_store.store_id,
  dim_store.store_address
```

```
SELECT
  dim_store.store_address,
  SUM(fact_sales.quantity) AS quantity_sold

FROM
  fact_sales
  INNER JOIN dim_product ON fact_sales.product_id = dim_product.product_id
  INNER JOIN dim_time ON fact_sales.time_id = dim_time.time_id
  INNER JOIN dim_store ON fact_sales.store_id = dim_store.store_id

WHERE
  dim_time.action_year = 2016
  AND dim_store.city = 'Berlin'
  AND dim_product.product_type = 'phone'

GROUP BY
  dim_store.store_id,
  dim_store.store_address
```

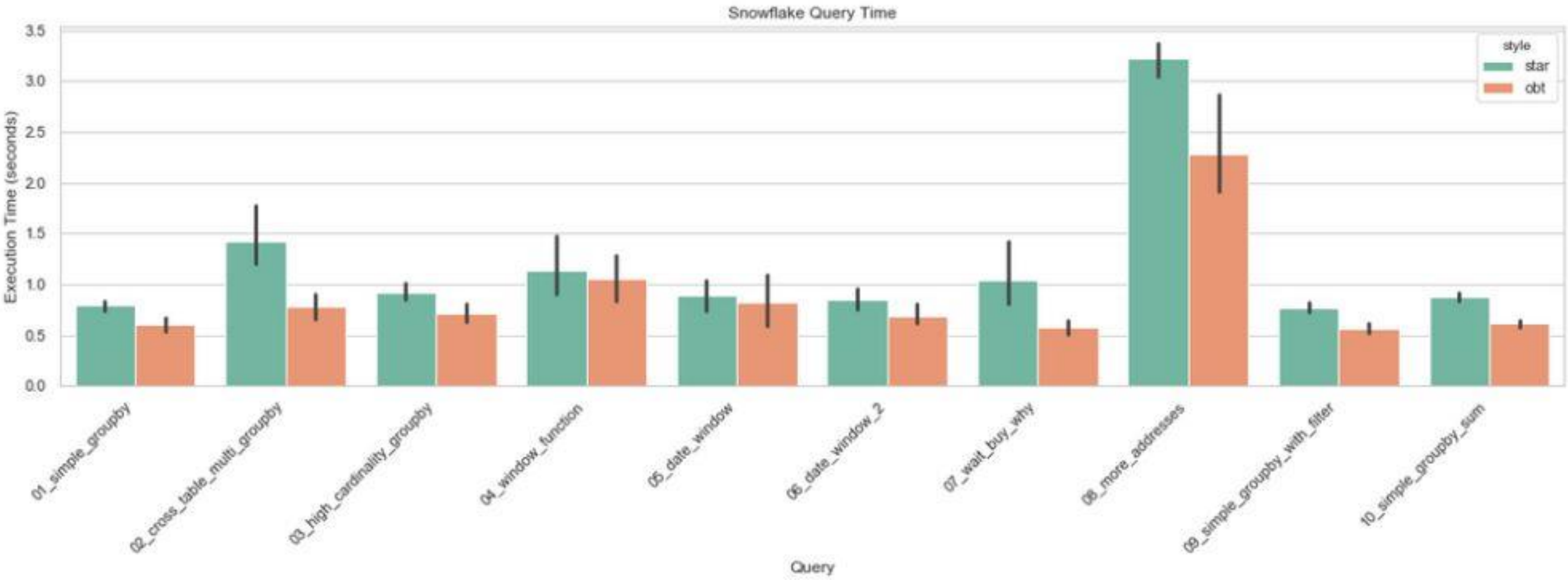
Reference: https://vertabelo.com/blog/data-warehouse-modeling-star-schema-vs-snowflake-schema/

What about "One Big Table"?



Reference: https://medium.com/dbsql-sme-engineering/one-big-table-vs-dimensional-modeling-on-databricks-sql-755fc3ef5dfd

What about “One Big Table”?



Reference: https://www.fivetran.com/blog/star-schema-vs-obt