



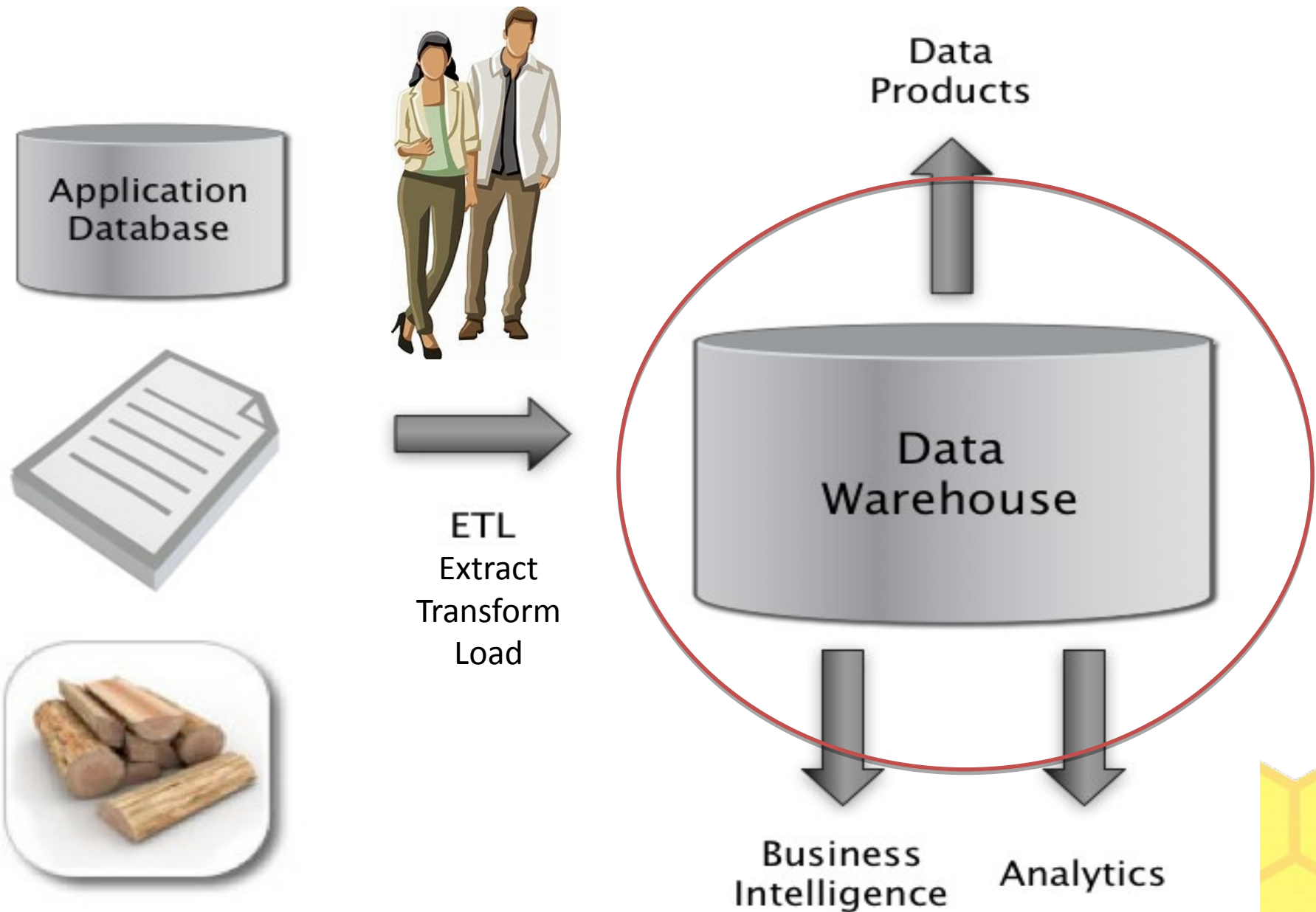
Workshop:Structured Query Language

Prepared by Associated Professor Krung Sinapiromsaran
Program director of Applied Mathematics and Computational Science,
Department of Mathematics and Computer Science,
Faculty of Science, Chulalongkorn University
25 June 2024, 13:00 – 16:00

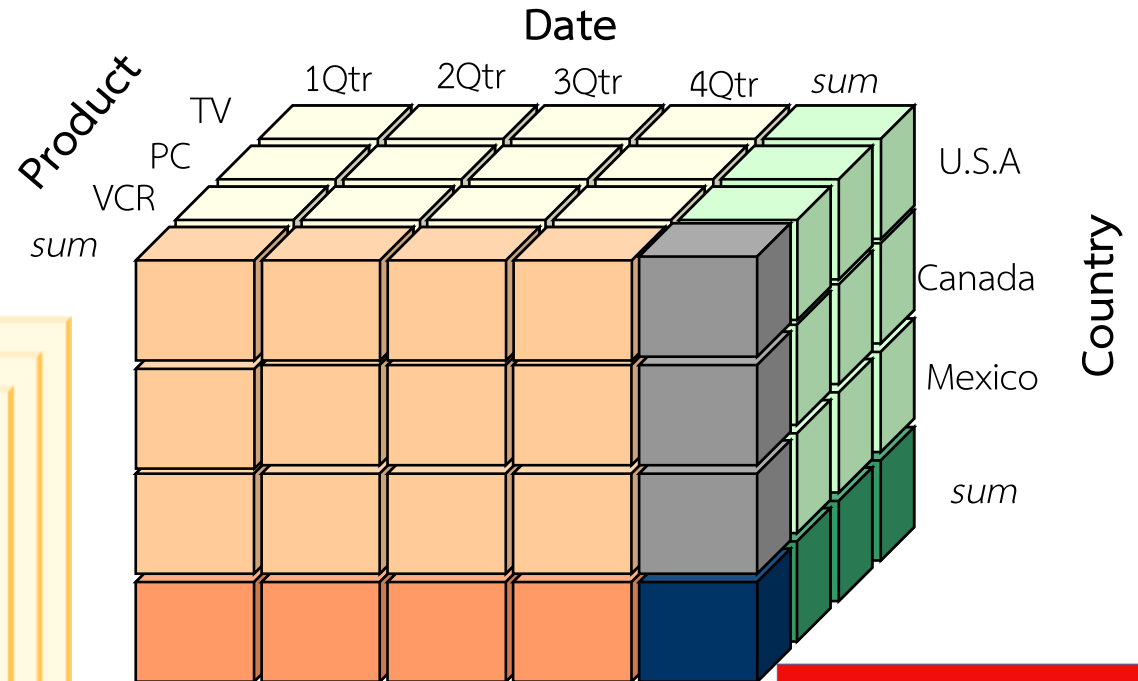
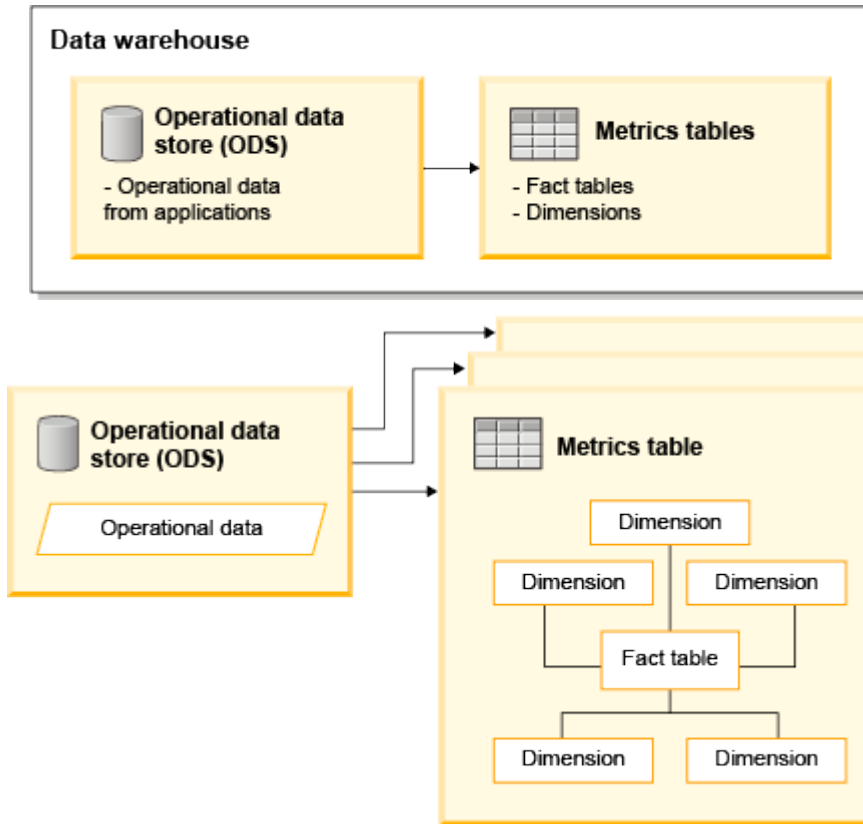
Contents

- Business Intelligence on Structured data
- Data model
- Relational algebra
- ACID property
- SQL = Structured Query Language
- Demonstration

Business Intelligence on structured data



MDDDB:Data warehouse

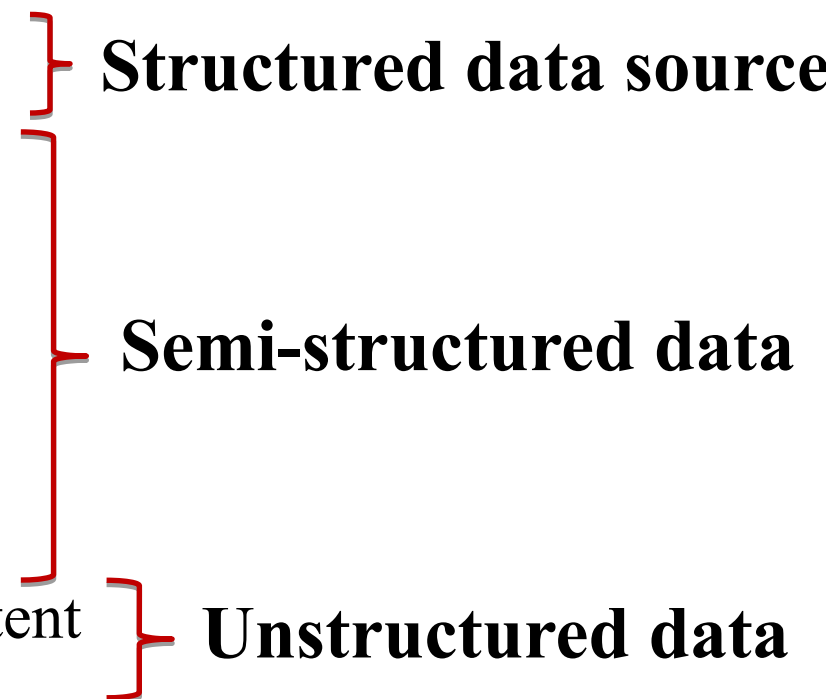


Grand total



Data model

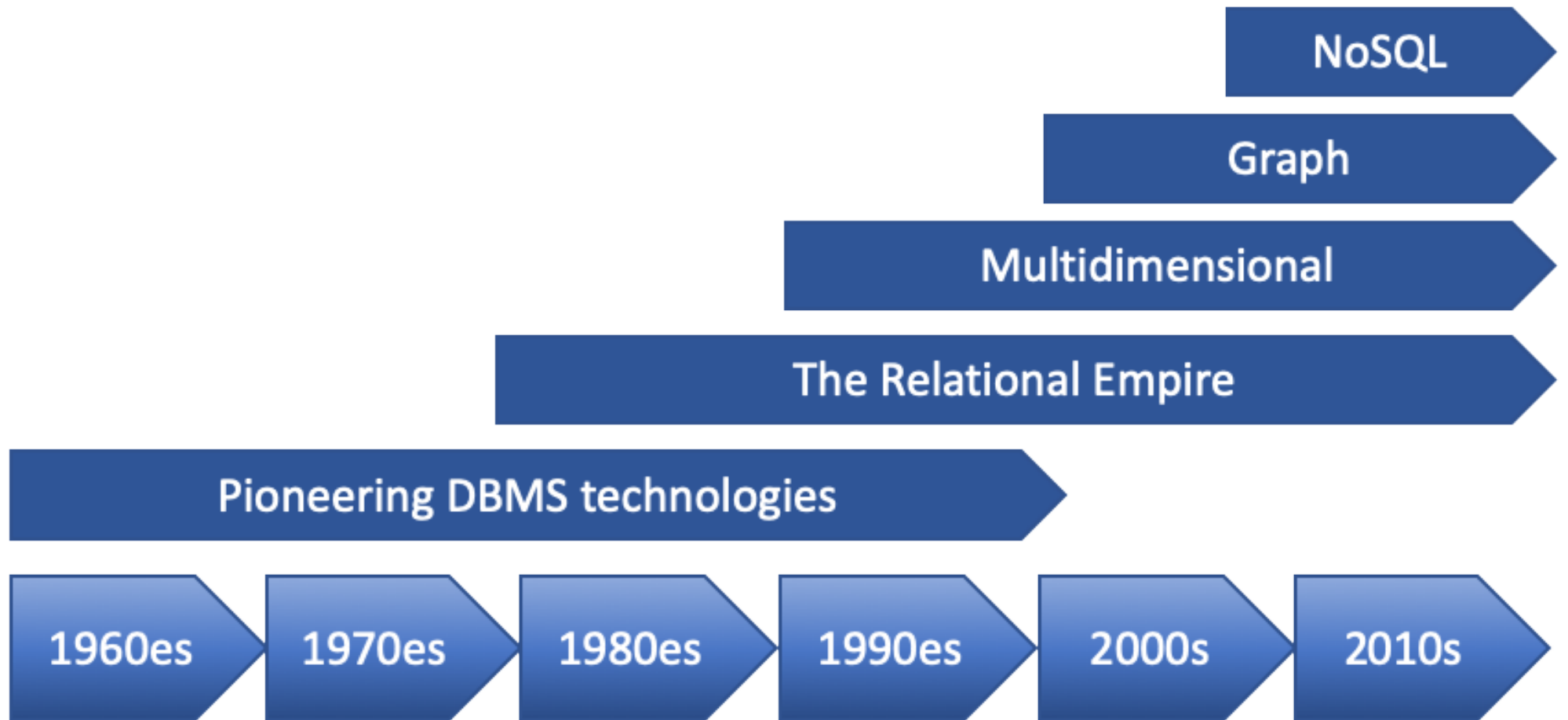
Various company data

- Company data sources
 - Application databases
 - Web server logs
 - Event logs
 - API server logs
 - Ad server logs
 - Search server logs
 - Advertisement landing page content
 - Images and video
 - Social media
 - Wikipedia
- Structured data source**
- Semi-structured data**
- Unstructured data**
- 
- A diagram on the right side of the slide categorizes the data sources. A red bracket groups 'Application databases', 'Web server logs', and 'Event logs' under the label 'Structured data source'. Another red bracket groups 'API server logs', 'Ad server logs', 'Search server logs', and 'Advertisement landing page content' under the label 'Semi-structured data'. A third red bracket groups 'Images and video' under the label 'Unstructured data'.

What is the data model? (CS)

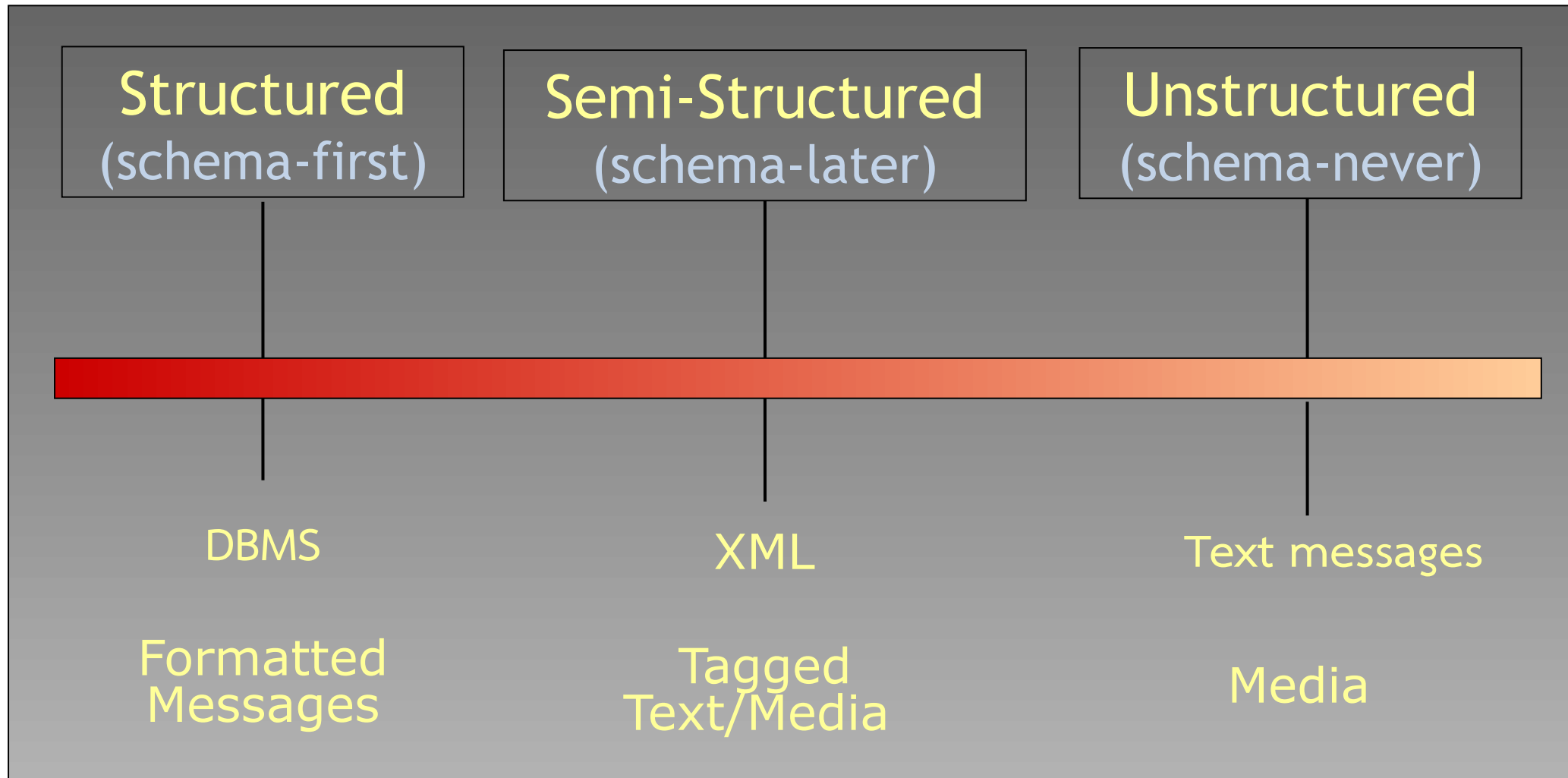
- A data model
 - is an abstract representation of data that defines its structure, organization, and relationships between different data elements;
 - acts as a blueprint of how data is organized and stored within a system or application.
- Three data types to be modeled
 - Structured data
 - Semi-structured data
 - Unstructured data

History of data model



Source: <http://graphdatamodeling.com/GraphDataModeling/History.html>

Structured Semi-Structured Unstructured



Structured data model

- **Structured data model** = adheres to a predefined and rigid format for organizing and representing data
 - **Relations** use in database management system. It organizes data into tables or relations, where each table represents an entity or relationship, and each row in the table represents an instance of that entity/relationship.

The relational model is based on mathematical set theory and predicate logic.

- **View as tables:** Data is stored in tabular form, with row representing individual entities and columns representing the attributes of those entities.

Semi-structured data model

- **Semi-structured data model** = a form of structured data that does not obey the tabular structure of data models associated with relational databases or other forms of data tables, but nonetheless contains tags or other markers to separate semantic elements and enforce hierarchies of records and fields within the data. Therefore, it is also known as self-describing structure.
 - **XML:** (Extensible Markup Language) a markup language and file format for storing, transmitting, and reconstructing arbitrary data. It defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.
 - **JSON:** (JavaScript Object Notation) is an open standard file format and data interchange format that uses human-readable text to store and transmit data objects consisting of attribute–value pairs and arrays (or other serializable values).

Unstructured data model

- **Unstructured data model** (or unstructured information) = information that either does not have a pre-defined data model or is not organized in a pre-defined manner. Unstructured information is typically text-heavy, but may contain data such as dates, numbers, and facts as well. This results in irregularities and ambiguities that make it difficult to understand using traditional programs as compared to data stored in fielded form in databases or annotated (semantically tagged) in documents.
 - **Natural language processing:** Data are collected as text corpora and been processed using either rule-based, statistical or neural-based approaches of machine learning and deep learning.
 - **Image:** Data are recorded as JPEG, PNG, and GIF. Most formats up until 2022 were for storing 2D images, not 3D ones.

Components of relational data model

- **Tables:** Tables are used to organize data into rows and columns. Each table has a unique name and consists of one or more columns, also known as attributes or fields.
 - **Relationships:** is defined how tables are related to each other.
- **Rows:** Rows, also called tuples, represent individual records or instances of data within a table. Each row contains values for each column defined in the table.
- **Columns:** Columns define the attributes or properties of the entities being represented. Each column has a name and a data type, which specifies the type of data that can be stored in that column.
- **Keys:** Keys are used to uniquely identify rows. A **primary key** is a column or combination of columns that uniquely identifies each row in the table. **Foreign keys** are used to establish relationships between tables.

Relational database: Example

- **Relation database:** a set of relations
- **Relation = Schema & Instances**

users (sid:INTEGER, name:NVARCHAR(10), login:NVARCHAR(20), age:INTEGER, gpa:REAL)

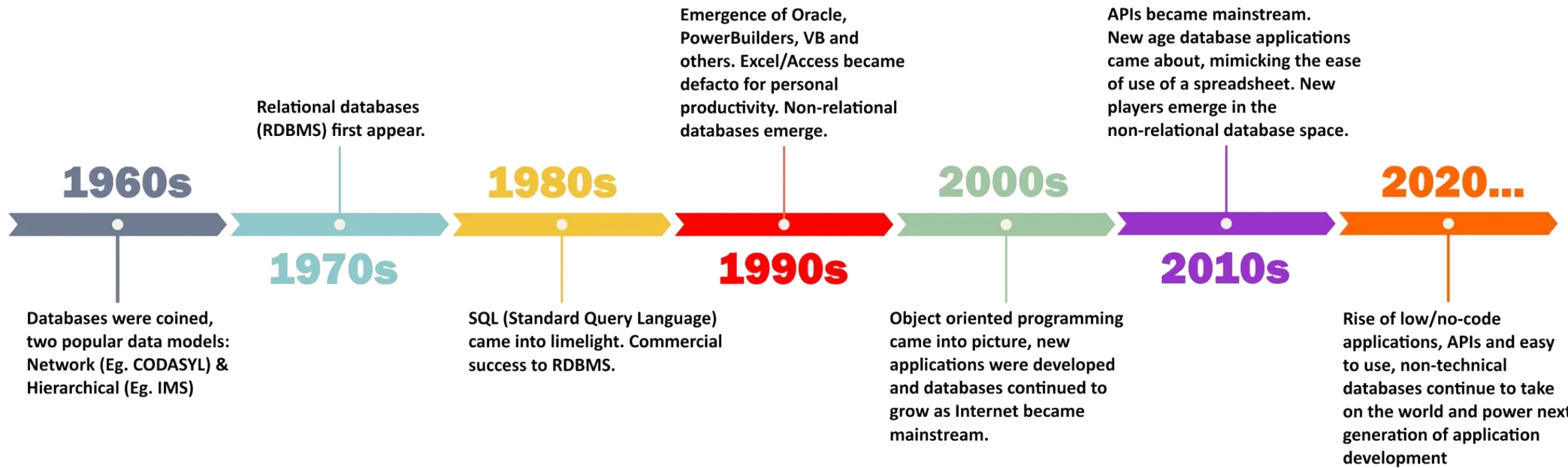
sid	name	login	age	gpa
53666	Jones	jones @cs	18	3.4
53688	Smith	smith@e e cs	18	3.2
53650	Smith	smith @m ath	19	3.8

- **users** relation has 5 arities
 - **sid** is a primary key, **name** contains text of each individual student, **login** contains login account, **age** keeps the age of each student, and **gpa** records the grade point average of each student.
 - The users relation has three records (students).



Relational algebra

History of Databases (1960-2020)

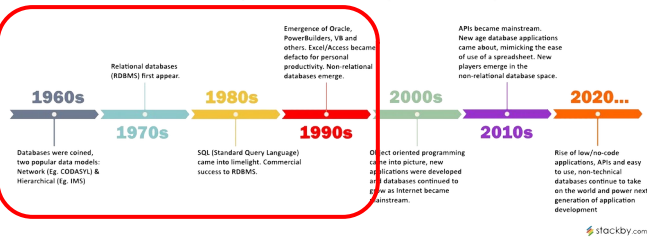


stackby.com

<https://dataxschool.medium.com/a-journey-through-time-the-evolution-of-databases-1960-2024-46ffd6db0b3f>

History of Databases

History of Databases (1960-2020)



Emergence of Oracle, PowerBuilders, VB and others. Excel/Access became defacto for personal productivity. Non-relational databases emerge.

1960s

Databases were coined, two popular data models: Network (Eg. CODASYL) & Hierarchical (Eg. IMS)

Relational databases (RDBMS) first appear.

1970s

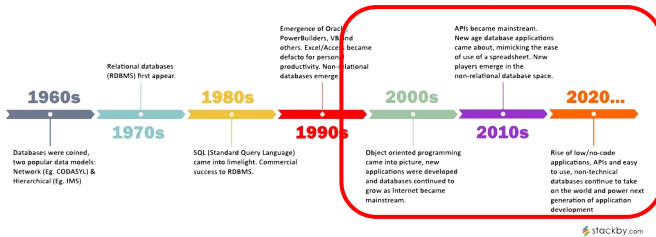
1980s

SQL (Standard Query Language) came into limelight. Commercial success to RDBMS.

1990s

History of Databases

History of Databases (1960-2020)



APIs became mainstream.

New age database applications came about, mimicking the ease of use of a spreadsheet. New players emerge in the non-relational database space.

2000s

Object oriented programming came into picture, new applications were developed and databases continued to grow as Internet became mainstream.

2010s

2020...

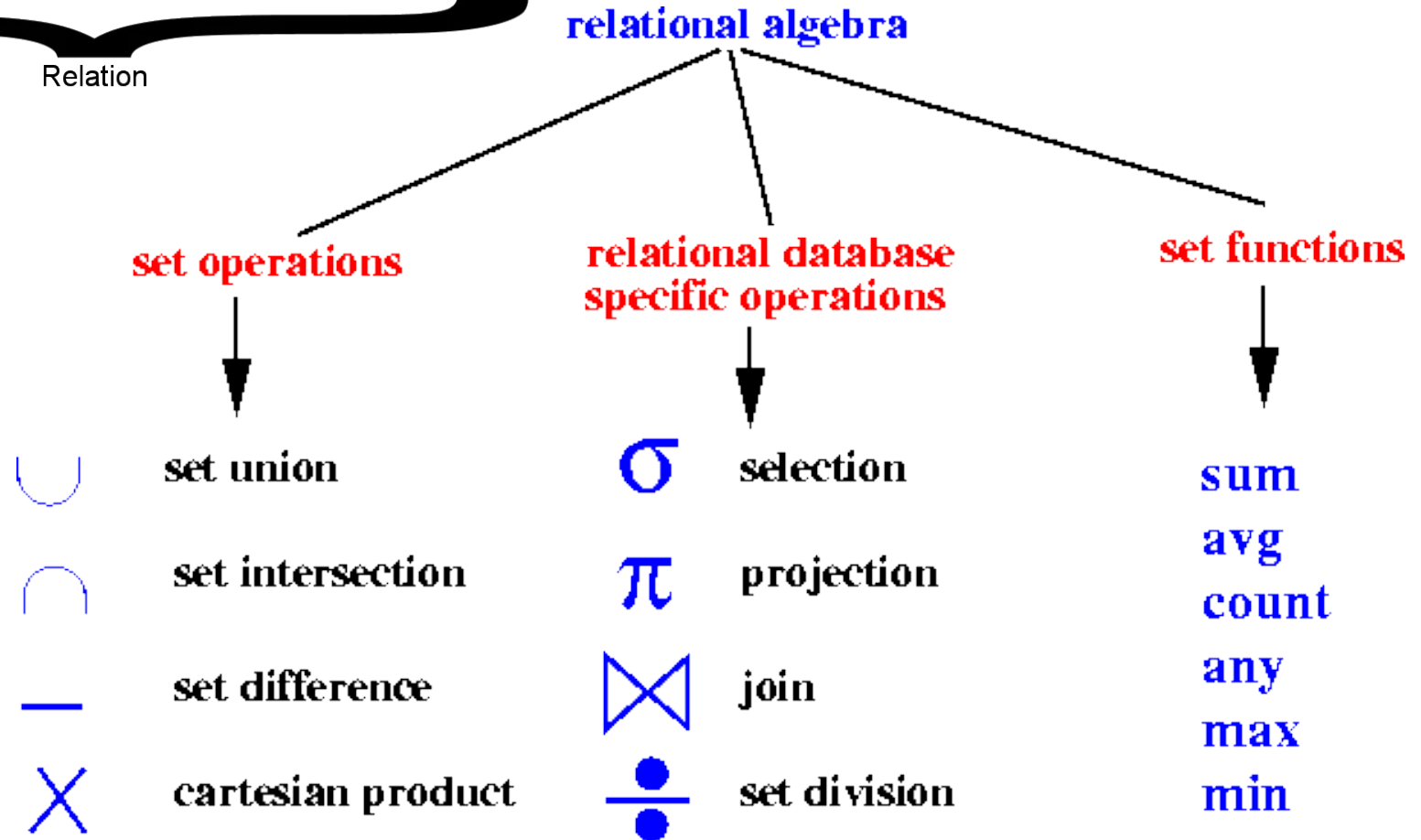
Rise of low/no-code applications, APIs and easy to use, non-technical databases continue to take on the world and power next generation of application development

Relational algebra

- **Relational algebra:** provides a set of operators to perform various operations on relations.
 - **Selection (σ):** selects tuples from a table based on a specific condition.
 - **Projection (π):** selects specific attributes from a table.
 - **Renaming (ρ):** renames attributes or relation.
 - **Union(\cup):** combines rows of two compatible tables.
 - **Intersection (\cap):** finds rows that exist in both tables.
 - **Set Difference ($-$):** find rows that exist in the first table but not in the second table.
 - **Cartesian Product (\times):** create a new table by combining all possible row combinations from two tables.

Attribute				
Tuple {				
Relation				

Relational algebra





Source: <https://www.codingalpha.com/acid-properties-in-dbms-and-sql/>

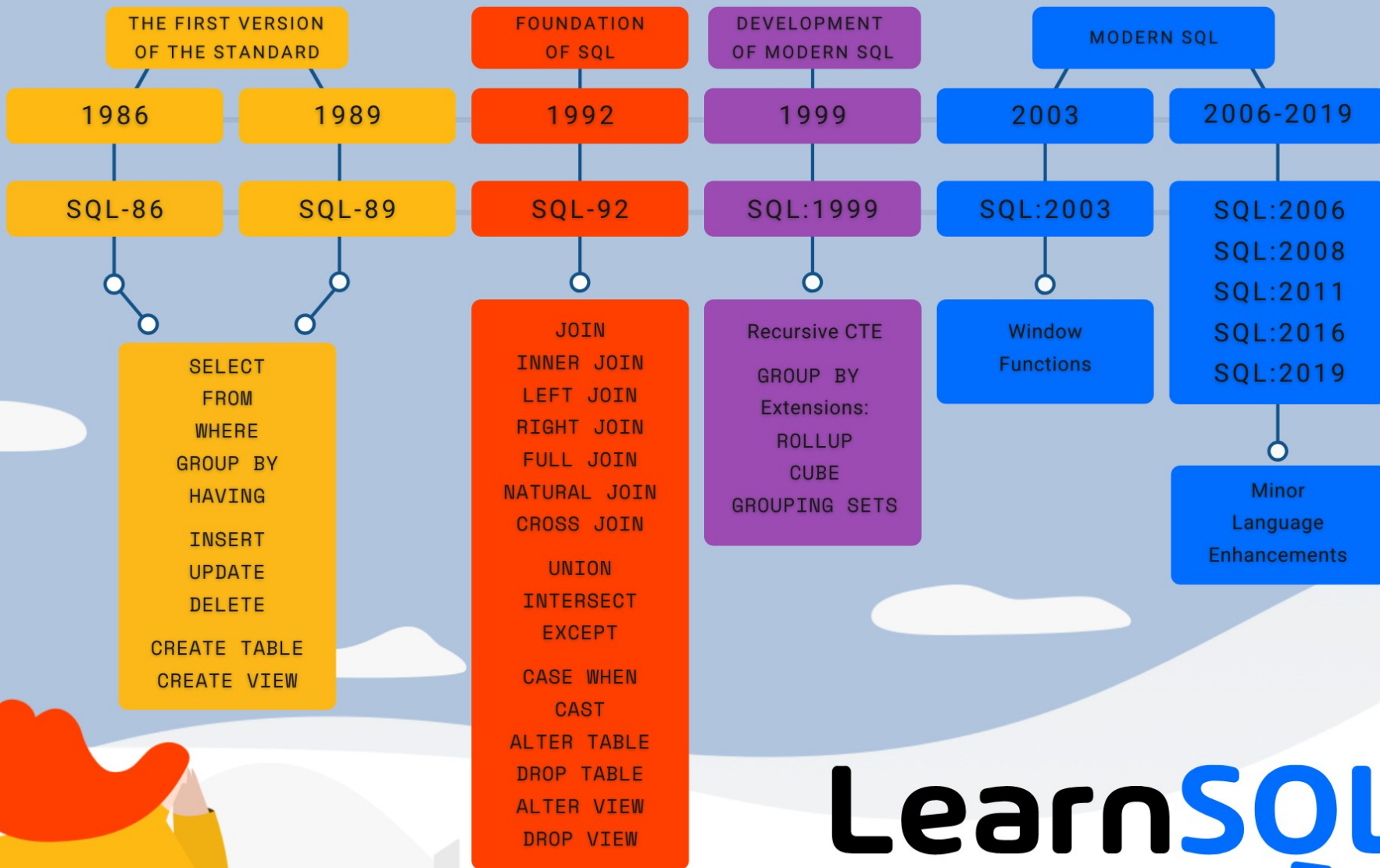
ACID property

- **ACID:** refers to four key properties that guarantee the reliability and validity of data, especially for transactions. ACID ensures that even if something goes wrong during a transaction processing, the data remains consistent and usable.
 - **Atomicity** ensures that all operations within a transaction are completed successfully. (all-or-nothing)
 - **Consistency** guarantees that only valid transitions are allowed. (data integrity rules and constraints)
 - **Isolation** ensures that concurrent transactions do not interfere with each other
 - **Durability** guarantees that once a transaction is committed (marked as successfully), the changes made to the database are permanent and persist even in the case of system failures.



SQL = Structured Query Language

The History of SQL Standards



LearnSQL
• com

Source: <https://learnsql.com/blog/history-of-sql-standards/>

SQL = Structured Query Language

- DDL = Data Definition Language is used to define and manipulate the structure of objects within a relational database
 - Functionality:
 - Create database objects (CREATE)
 - Modify database objects (ALTER)
 - Delete database objects (DROP)
- DML = Data Manipulation Language is used to extract and update tuples

DDL

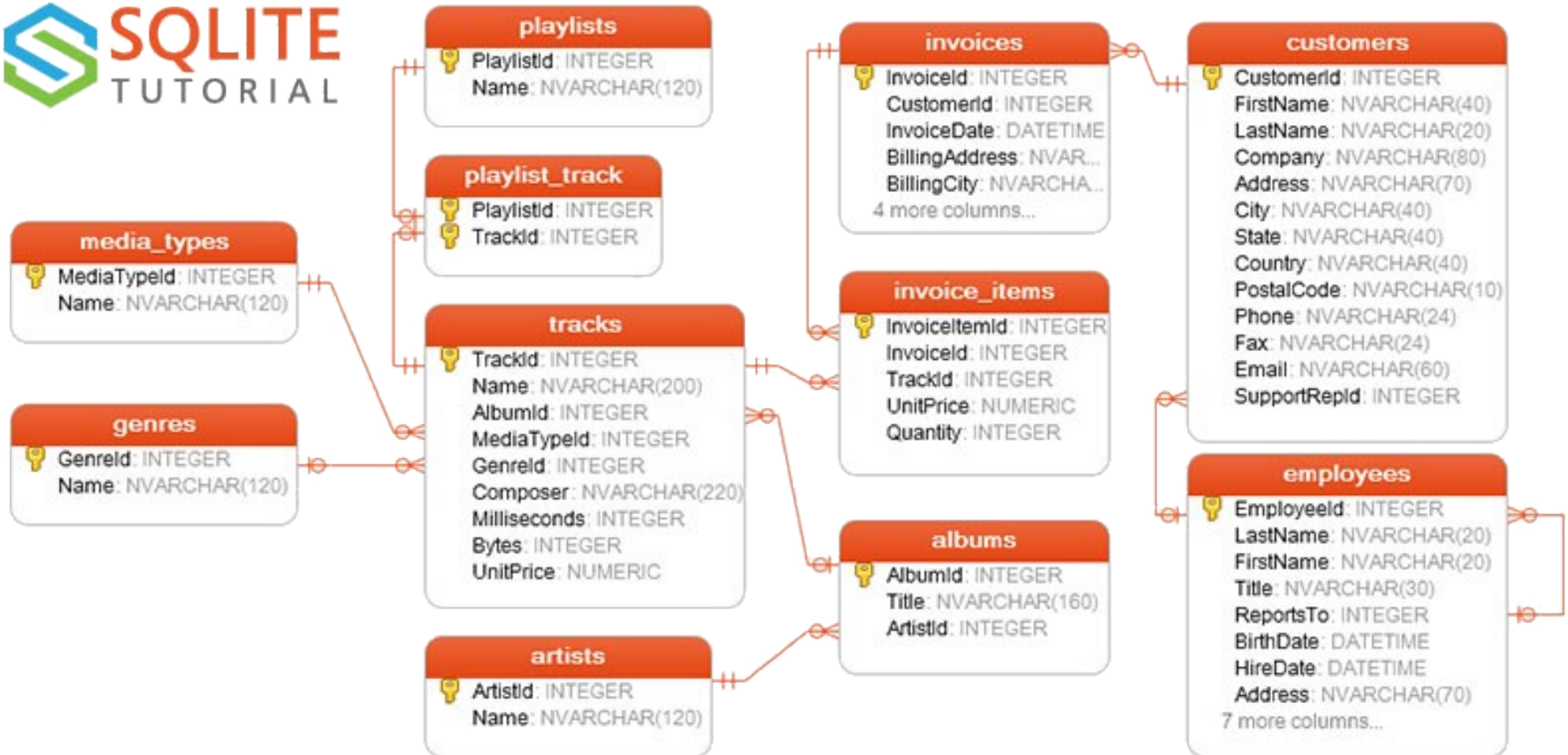
- CREATE TABLE relation_name (a1: type, a2: type, ..., ad:type) = create a new relation by relation_name with given attributes.
- INSERT INTO relation_name (a1, a2, ..., ad) VALUES (v1, v2, ..., vd)
- DELETE FROM relation_name WHERE condition = remove tuples that satisfies condition

* Developed at [IBM](#) by [Donald D. Chamberlin](#) and [Raymond F. Boyce](#) in the 1970s.
Used to be *SEQUEL* (*Structured English QUery Language*)



Demonstration

Chinook ER diagram



Structured query language

SELECT	[DISTINCT] <i>target-list</i>
FROM	<i>relation-list</i>
WHERE	<i>qualification</i>
	[Extension]

- SELECT (projection operation) target-list is a list of fields to be projected.
- FROM relation-list is a list of relations
- WHERE qualification
- DISTINCT for selecting values that are unique
- Extension can be
 - LIMIT n = showing only n tuples
 - ORDER BY attribute-list [ASC|DSC] sorting ascendingly or descendingly

Preparation for SQL

```
# Import sqlite library  
import sqlite3
```

```
# Get the chinook database via sqlitetutorial  
!wget http://www.sqlitetutorial.net/wp-content/uploads/2018/03/chinook.zip  
!unzip chinook.zip
```

```
# Create the database connection with the current chinook.db  
dbconnect = sqlite3.connect('chinook.db')
```

```
# Use cursor to connect to DB  
cursor = dbconnect.cursor()
```

Show tuples using SQL

```
# Show each record in employees table  
cursor = dbconnect.cursor()  
cursor.execute("SELECT * FROM employees")  
[row for row in cursor.fetchall()]
```

```
# Extract the field names  
names = [description[0] for description in cursor.description]  
print(names)
```

```
# To extract the type, need to get it from pragma_table_info()  
cursor.execute("SELECT name, type FROM pragma_table_info('employees')")  
[row for row in cursor.fetchall()]
```


Projection & Selection

```
# Show two columns: firstname and lastname in employees table  
cursor.execute("SELECT firstname, lastname FROM employees")  
[row for row in cursor.fetchall()]
```

```
# Only distinct result can be shown  
cursor.execute("SELECT DISTINCT title FROM employees")  
[row for row in cursor.fetchall()]
```

```
# Pick from invoice_items table only trackid that have unitprice equal to 1.99  
cursor.execute("SELECT trackid FROM invoice_items WHERE unitprice == 1.99 LIMIT 10")  
[row for row in cursor.fetchall()]
```


Rename & Join

Any attribute can be renamed using AS

```
cursor.execute("SELECT name AS TrackName FROM tracks LIMIT 10")
```

```
[row for row in cursor.fetchall()]
```

Show 10 records representing which artists is the owner of which albums

```
cursor.execute("SELECT Title, Name FROM albums A, artists T WHERE A.artistid ==  
T.artistid LIMIT 10")
```

```
[row for row in cursor.fetchall()]
```

Alternatively, join should be used

```
cursor.execute("SELECT Title, Name FROM albums A INNER JOIN artists T ON A.artistid  
== T.artistid LIMIT 10")
```

```
[row for row in cursor.fetchall()]
```

Aggregation

Show the number of customers records

```
cursor.execute("SELECT count(*) AS 'Total customer' FROM customers")  
print(cursor.fetchone())
```

Show the total of quantity purchase

```
cursor.execute("SELECT sum(Quantity) AS 'Total quantities' FROM invoice_items")  
print(cursor.fetchone())
```

Show the average number of milliseconds in tracks table

```
cursor.execute("SELECT avg(milliseconds) AS 'Average tracks' FROM tracks")  
print(cursor.fetchone())
```

Show the maximum millisecond in tracks table

```
cursor.execute("SELECT max(milliseconds) FROM tracks")  
print(cursor.fetchone())
```

Show the maximum millisecond in tracks table

```
cursor.execute("SELECT min(milliseconds) FROM tracks")  
print(cursor.fetchone())
```

Nested query

Show the minimum millisecond in tracks table

```
cursor.execute("SELECT * FROM tracks, (SELECT min(milliseconds) AS mintrack  
FROM tracks) M WHERE milliseconds == M.mintrack")  
print(cursor.fetchone())
```

Show the maximum millisecond in tracks table

```
cursor.execute("SELECT * FROM tracks, (SELECT max(milliseconds) AS maxtrack  
FROM tracks) M WHERE milliseconds == M.maxtrack")  
print(cursor.fetchone())
```

Derived field & Group by

```
# Compute the payment
cursor.execute("SELECT DISTINCT unitprice*quantity AS 'payment' FROM invoice_items")
[row for row in cursor.fetchall()]
```

```
# Aggregate them
cursor.execute("SELECT sum(unitprice*quantity) AS 'Total payment' FROM invoice_items")
print(cursor.fetchone())
```

```
# Generate table using group-by
cursor.execute("SELECT customerID, count(*), sum(Total) AS T FROM invoices GROUP BY
CustomerID HAVING T > 40")
[row for row in cursor.fetchall()]
```