1 Database design : normalization

Goals of a good data structure design

- **Data Integrity**: consistency, accuracy, avoiding anomalies
- **Query performance**: fast data retrieval
- Allow for future **expansion** of data types or relationships, evolution business requirements
- **Scalability** : growth in data volume
- **Storage** : minimize data redundancy (also important for integrity)

and :

- **Simplicity**: Create an understandable structure for developers and analysts

It always comes down to balancing between read and write performance
How to design the data structure of a database ?

2 Some background and recap

quick recap:

- Entity-Relationship Diagrams : ERDs
- 1-1, 1-many, many-many relations

and then:

- OLAP vs OLTP databases and design strategies
- Normalization
- anomalies to detect the need for normalization
- normalization criteria: normal forms: 1NF, 2NF, 3NF
- Denormalization when needed (OLAP)
- Functional dependency

Practice:

- we'll normalize the trees v01 database

---

3 Entity Relation Diagrams

A dude called Peter Chen developed the ER diagram in 1976.

The ER model was created to visualize data structures and relationships in many situations.

- Object-Oriented Systems
- software architecture
- Business Process Modeling
- data flow in various systems
- relational databases

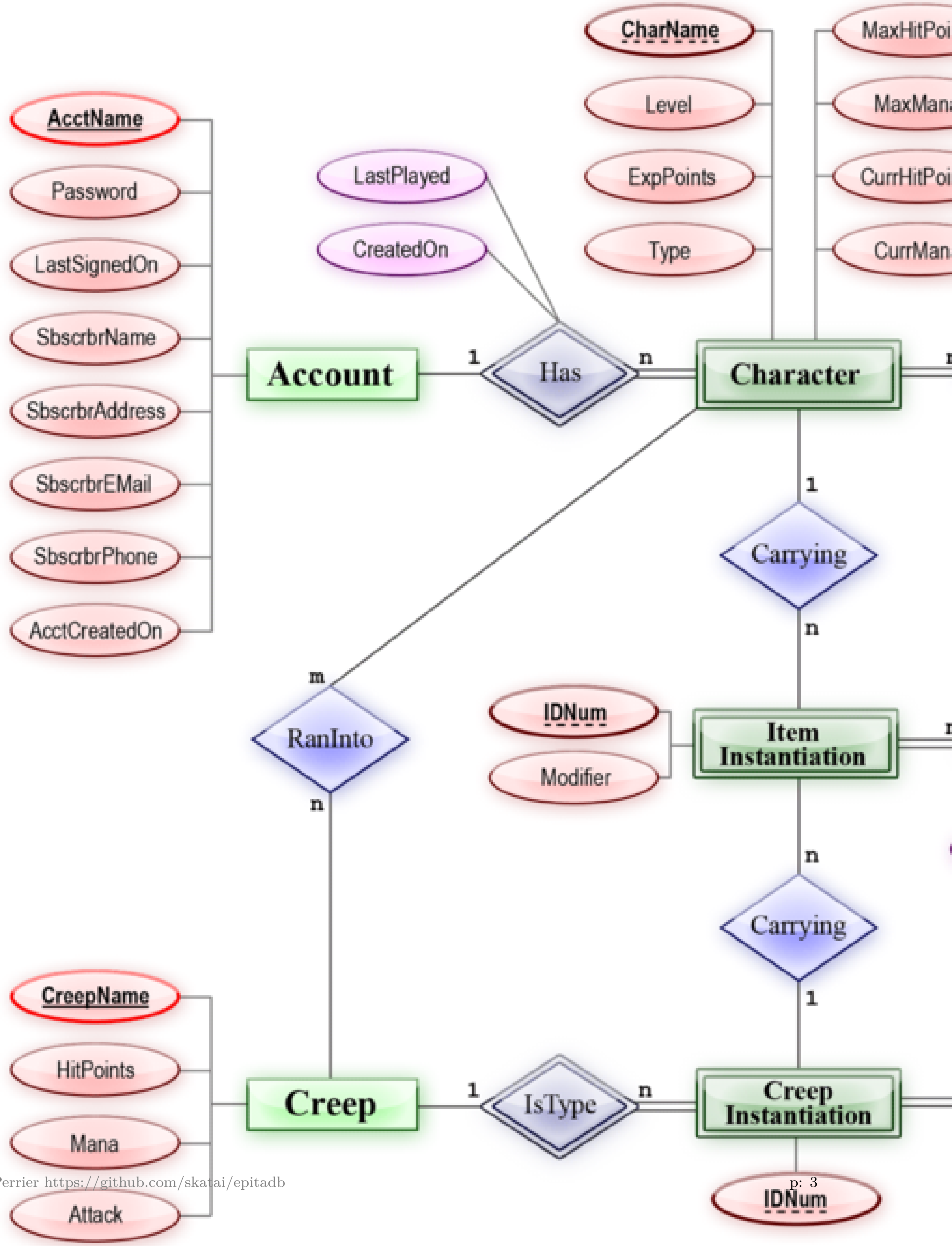
ER diagrams help in system design, information architecture, and application workflows.
The components of an ER diagram are :

- entities (tasks, real world object, …)
- attributes
- relations between the entities

See this article for a complete explanation of ER diagrams. As you can see there are many types of entities and attributes : strong, weak, key, composite, etc …

Introduction of ER model

see also the wikipedia page


CharName
Level
ExpPoints
Type
AcctName
Password
LastSignedOn
SbscrbrName
SbscrbrAddress
SbscrbrEMail
SbscrbrPhone
AcctCreatedOn
LastPlayed
CreatedOn
Account
Has
Character
Carrying
RanInto
IDNum
Modifier
Item Instantiation
Carrying
CreepName
HitPoints
Mana
Attack
Creep
IsType
Creep Instantiation
IDNum
1
n
1
n
m
n
n
1
1
n

3.1 ER diagram for relational databases

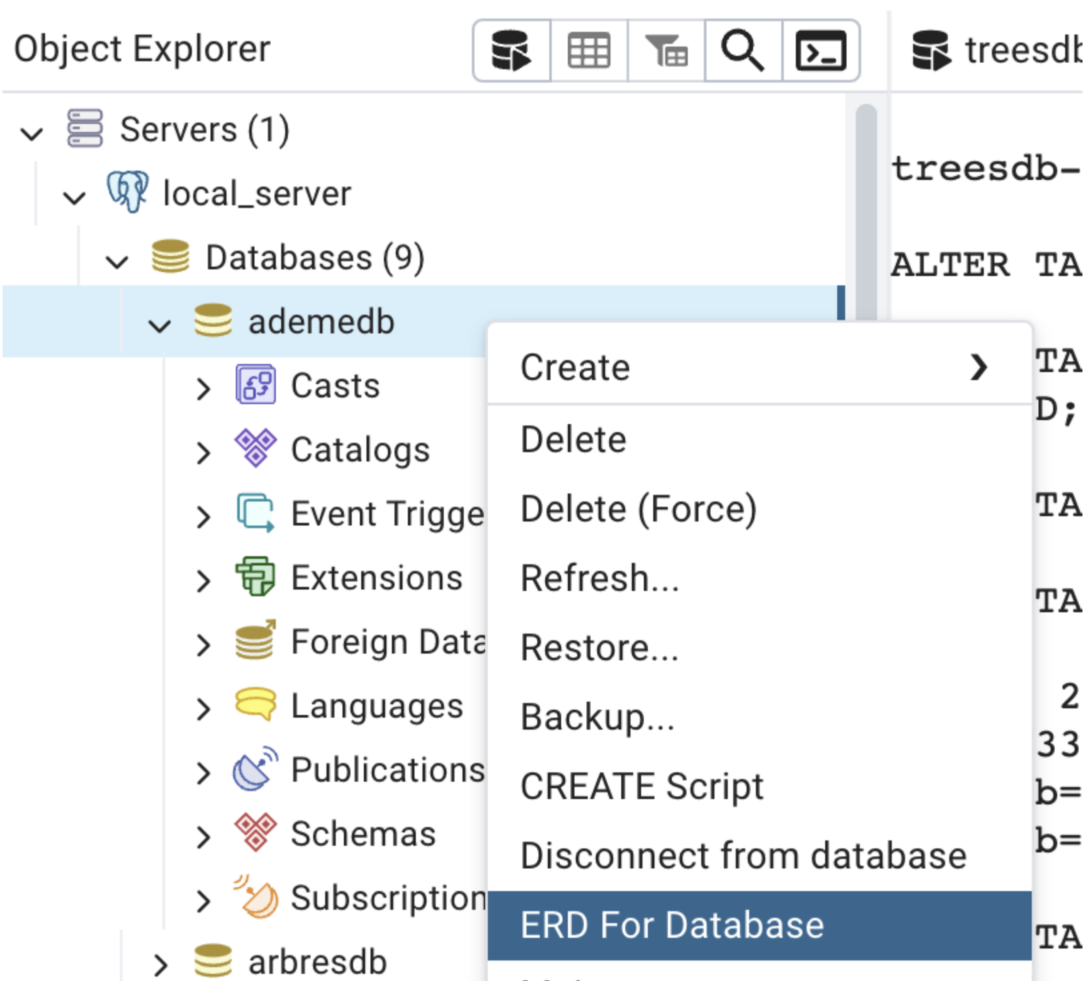
For databases, **the ER Diagram represent the structure of the database.**

- entities are **tables**
- attributes are table **columns**
- **relations** between entities can be
  - one to one
  - one to many
  - many to many

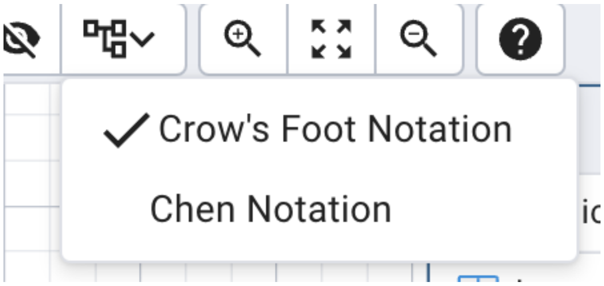
The ER diagram displays the **relations** between the **entities** (tables) present in the database and lists their **attributes** (columns)

3.2 Generate and ERD in pgAdmin

- connect to the remote server on the airdb database
- click right on the database name
- click on ERD for database



You can change notation for the relation type with



---

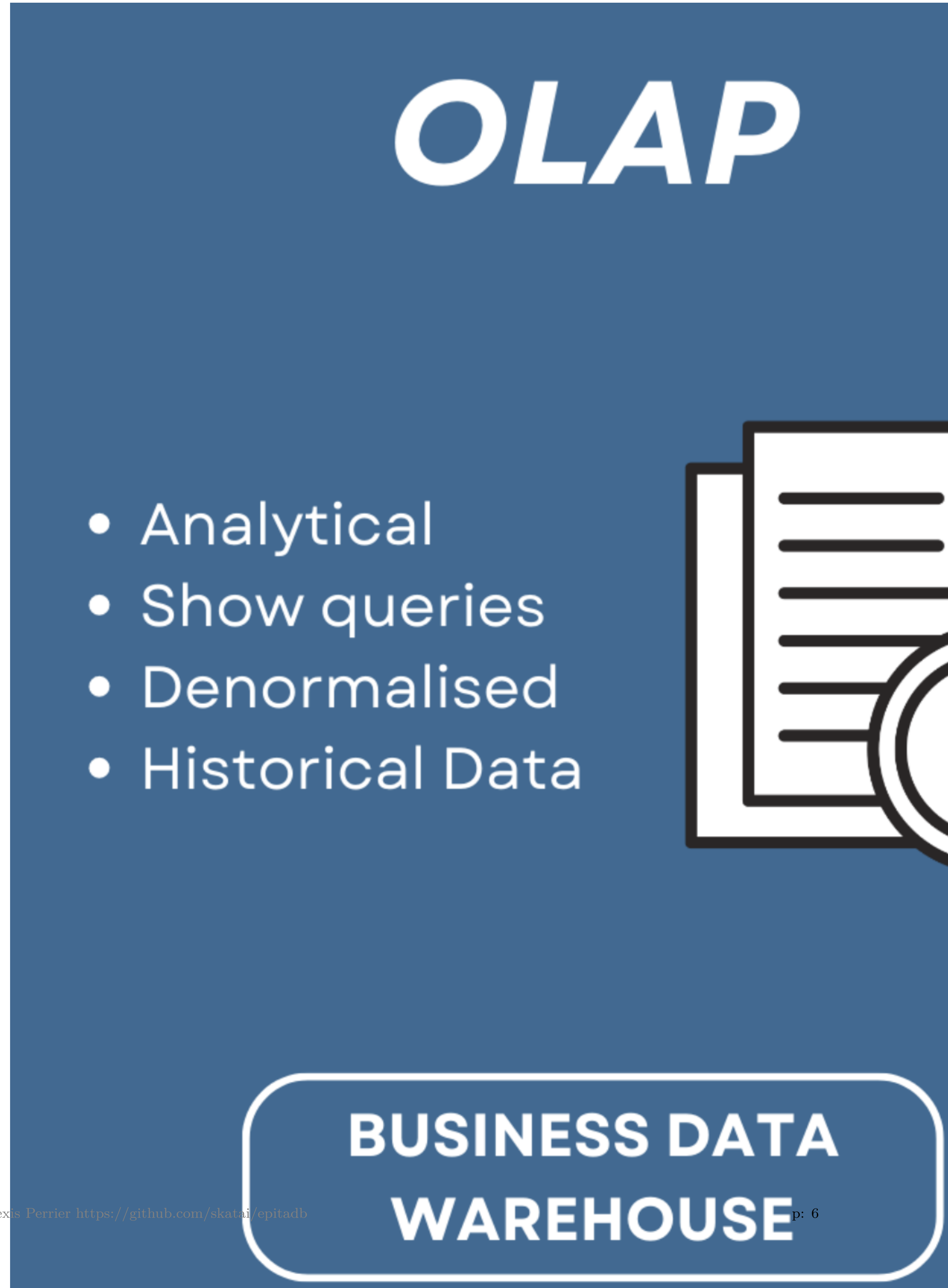
4 OLAP vs OLTP

The end usage of the database drives its data structure

analytical databases vs **transactional** databases

OLAP : Online Analytical Processing - analysis, BI, reporting, dashboards, - optimized for high read volume - complex queries (lots of joins and calculations) which have to be somewhat fast - can be asynchronous, query execution does not have to be lightning fast

OLTP: Online Transaction Processing, - applications, transactions, high write volume - optimized for high write volume: **data integrity**, fast updates and inserts - ACID properties for transactions (all or nothing) (ACID: (Atomicity, Consistency, Isolation, Durability)) - synchronous, real time


OLAP
• Analytical
• Show queries
• Denormalised
• Historical Data
BUSINESS DATA
WAREHOUSE

- Further reading (look at the difference table and the Q&A at the end of the article) : difference between olap and oltp in dbms

4.1 Quiz

For each scenario, determine whether it's more suited for an OLTP (Online Transaction Processing) or OLAP (Online Analytical Processing) system.

- Liking a friend's post on Instagram.
- Analyzing trending hashtags on Twitter over the past month.
- Sending a Snapchat message to a friend.
- Netflix recommending shows based on your viewing history.
- Ordering food through a delivery app.
- Making an in-app purchase.
- TikTok or Youtube analyzing which video types keep users watching longer.
- A fitness app calculating your average daily steps for the past year.

solution

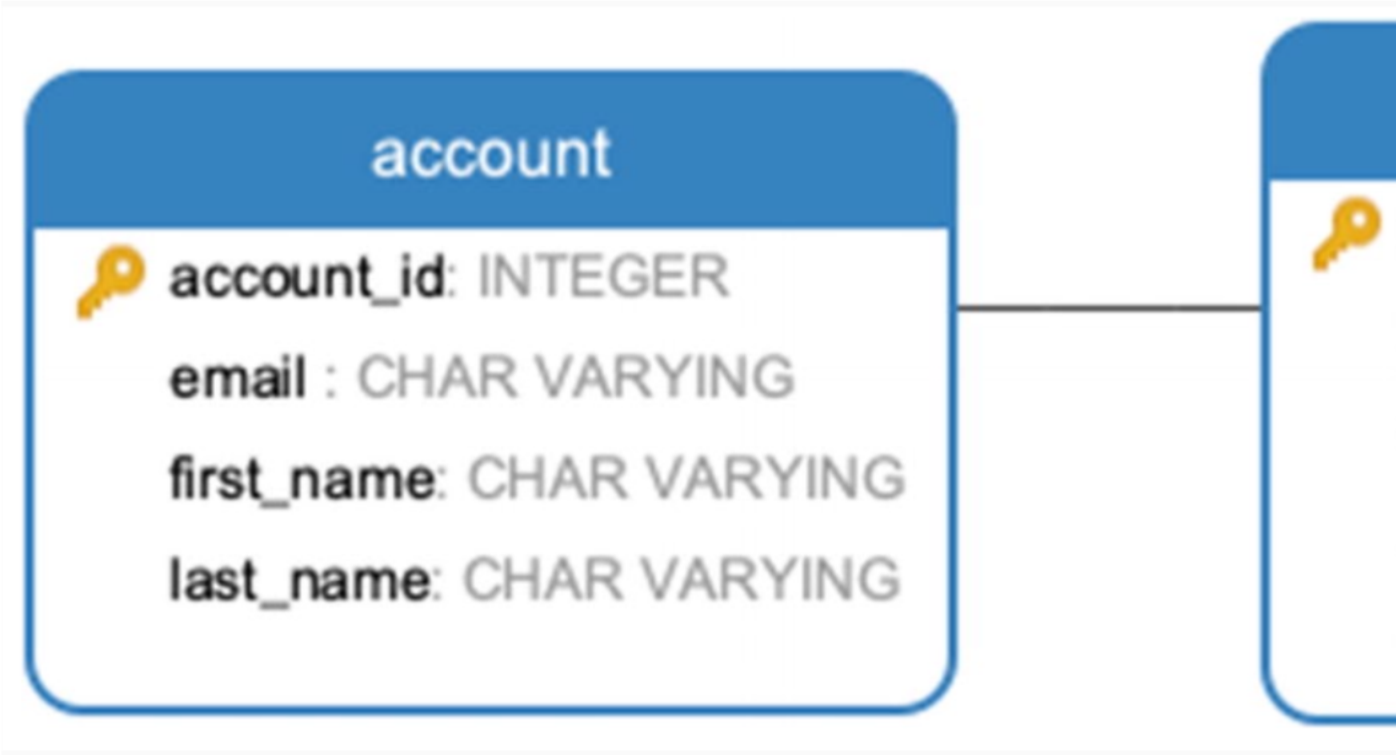
your answers

---

5 Choosing between 2 designs

1 account table with multiple phones

1 account table and 1 dedicated phone table



which design (1 or 2 tables) is betterin terms of faster or simpler query for:

- fast retrieval search over phone number(s)
- dealing with missing phone type
- adding a new phone type
- flagging a phone as primary
- handling a user with no phone
- displaying all the phones of an account in a UX

---

6 Normalization

The general goal of **normalization** is to reduce data **redundancy** and **dependency** by organizing data into **separate, related tables**.

This helps maintain data integrity and flexibility:

- logical entities
- independence between tables
- uniqueness of data

Normalized databases are

- easy to update
- easy to maintain

Informally, a database is normalized if all column values depend only on the table primary key, and data is decomposed into multiple tables to avoid repetition.

In the *1 table design* for the account and its phone numbers, a phone number value depends on the name of the phone column (home_phone, work_phone, …) not just the account_id key : it's not normalized

With a dedicated phone table, the phone value depends only on the phone_id key : normalized

---

7 Denormalization

The idea of denormalization is to have data redundancy to simplify queries and make OLAP queries faster

Redundant data : the same data / info exists in multiple tables

select queries may involve less joins but updates are more complex and data integrity is more complex to preserve.

7.1 Scenario:

In a social network, you have two tables:

1. **Users table**: Contains user information like `user_id`, `name`, and `email`.
2. **Posts table**: Contains posts made by users, with fields like `post_id`, `user_id`, and `content`.

In a **normalized** database: the `Posts` table only contains `user_id` as a foreign key.

If if you want to display the user's name next to their post, you need to **JOIN** `Users` and `Posts` tables.

To improve performance you can **denormalize** the Posts table by adding the `user_name` to the `Posts` table.

Denormalized `Posts` table:

post_id	user_id	user_name	content
1	101	Ulaf	Hello world!
2	102	Birgitte	Loving the sun
3	103	Inge	Great day!
4	114	Boris	When's the break?

- Faster read performance: You can fetch the `user_name` along with the post data without needing to perform a join between the `Users` and `Posts` tables.

But

- **Data redundancy**: If Ulaf changes his name, you will need to update it in both the `Users` table and every row in the `Posts` table that references him. This increases the complexity of updates.

---

8 Anomalies

Given a database, how do you know if it needs to be normalized ?

There are 3 types of anomalies that you can look for

- insertion
- update
- deletion

normalization would solve these anomalies.

8.1 Insertion anomalies

Consider the table of teachers and their courses

id	teacher	course
1	Bjorn	Intermediate Database
2	Sofia	Crypto Currencies
3	Hussein	Data Analytics

Now a new teacher (Alexis), is recruited by the school. The teacher does not have a course assigned to yet. If we want to insert the teacher in the table we need to put a `NULL` value in the course column.

and NULL values can cause problems (more on that later)

More generally:

- in a relation where an item (A) has many (or has one) other items (B), but A and B are in the same table.
- so for a new item A without items B, inserting a new A without items B means the value for B has to be null

8.2 Update anomaly

Consider now the following movies, directors, year and production house

Film_ID	Film_Name	Director_Name	Release_Year	Production_House
1	Sholay	Ramesh Sippy	1975	Sippy Films

Film_ID	Film_Name	Director_Name	Release_Year	Production_House
3	Kabhi Khushi Kabhie Gham	Karan Johar	2001	Dharma Productions
4	Kuch Kuch Hota Hai	Karan Johar	1998	Dharma Productions
5	Lagaan	Ashutosh Gowariker	2001	Aamir Khan Productions
6	Dangal	Nitesh Tiwari	2016	Aamir Khan Productions
7	Bajrangi Bhaijaan	Kabir Khan	2015	Salman Khan Films
8	My Name Is Khan	Karan Johar	2010	Dharma Productions

Film_ID	Film_Name	Director	Release_Year	Production_House
[illegible]	Gully Boy	Zoya Akhtar	2019	Excel Entertainment
[illegible]	Zindagi Na Milegi Dobara	Zoya Akhtar	2011	Excel Entertainment

(chatGPT knows Bollywood)

If one of the production house changes its name, we need to update multiple rows

In a small example like this one this is not a problem since the query is trivial but in large databases some rows may not be updated

- **human error** when writing the update query or doing manual updates
- `**Redundant Data**: updating multiple rows _at the same time_  can cause performance issue`
- `**Data Locking**: In a multi-user environment, updating several rows at once may lock part`
- `if changes are made in stages or **asynchronously**, data can temporarily or permanently e`

Moving the production house data in its own separate table, updating its name would only impact one row!

8.3 Deletion errors

Here is a table about programmers, languages and projects

() () () ()
* * * *
DeDeLan-
0.3971Project_Name
0.1618 velguage
0.2353 oper_IDName
0.20.2353oper_IDName

() () () ()
* * * *
0.20.23161387 Atomic
Blaster

() () () ()
* * * *
0.20.29261387 Java In-
vis-
i-
ble
tank

() () () ()
* * * *
0.20.23161387 Atomic
gomery Blaster

() () () ()
* * * *
0.20.294613871 JavaScript Ex-
gie plod-
ing
Squir-
rel

() () () ()
* * * *
0.20.2956188C++ War
AI

() () () ()
* * * *
0.20.2966158 Atomic
Blaster

() () () ()
* * * *
0.20.2376138 War
AI

If for imperative reasons we want to scrape all records of the *War AI* project from the database, and we delete all rows that mention that project then we also delete Marge and Ned if they are not involved in other projects

8.4 In short

When you have different **natural / logocal entities** packed together in the same table, there will be anomalies and you need to normalize.

---

9 The problem with null values

Why is it a problem to have NULL values in a column?

- **Ambiguity**: Null can mean "unknown," "not applicable," or "missing," leading to confusion.
- requires **special handling** and complicates
  - **queries**: (`IS NULL` vs `=`) .
  - **data analysis**: `NULL` values are ignored in aggregate functions (like `SUM`, `COUNT`).
- Impacts **indexing and performance**: since Nulls are excluded from indexes
- **Violates normalization**: indicates poor database design or incomplete relationships.

So avoid NULL values!

---

10 What about anomalies in the trees table ?

What anomalies can you find for each type: insertion, update and deletion

Insertion Anomalies: - To add a new tree species or genre without an associated tree, we'd have to insert a record with null values for most fields, which is not ideal. - We can't add a new location without adding a tree, even if no trees exist there yet.

Update Anomalies: - If we need to update information about a species (e.g., its genre), we'd have to update multiple rows, risking inconsistency. - Updating location information (e.g., arrondissement name) would require updating multiple rows, again risking inconsistency.

Deletion Anomalies: - If we delete the last tree of a particular species, we lose all information about that species. - Deleting all trees in an arrondissement would also delete all information about that arrondissement.

---

11 In short

Need for normalization when:

- same data exists in multiple rows
- multiple columns with data of same nature, different types
- same data exists in multiple tables

or simply said:

11.1 every non-key attribute must provide a fact about the key, the whole key, and nothing but the key

---

12 Normal forms

A normal form is a **rule** that defines a level of normalization.

In general a database is considered normalized if it meets 3NF level.

- UNF: Unnormalized form
- 1NF: First normal form
- 2NF: Second normal form
- 3NF: Third normal form

But there are multiple other levels

- EKNF: Elementary key normal form
- BCNF: Boyce–Codd normal form
- 4NF: Fourth normal form
- ETNF: Essential tuple normal form
- 5NF: Fifth normal form
- DKNF: Domain-key normal form
- 6NF: Sixth normal form

It gets very abstracts very quickly.

Normal forms are a gradual step by step process towards normalization. Each is a guide that focuses on a single type of problem. We can always choose to apply a normalization form or to ignore it

In the following, we mention about:

- Relation : table
- Attribute : column
- Primary key
- Composite key : key composed of multiple attributes

13 1NF

Each field contains a single value

> A relation is in first normal form if and only if no attribute domain has relations as elements.

In short: columns cannot contain relations or composite values : arrays, json, other tables

More on 1NF : Wikipedia: Satisfying 1NF

- Does the tree table follow 1NF ?

13.1 Wait, ... what ?

There's a contradiction between first normal form (1NF) and the existence of **ARRAY** and **JSONB** (see also POINT, HRANGE, composite types, and many other types) data types in SQL databases

The rule of thumb is :

When you frequently need to access, modify, handle the elements of the sets of values for a given record then> apply 1NF

It's a balance between simplicity and control
For instance:

- Use normalization (1NF) when:
  - Storing a list of order items, where each item needs its own price, quantity, and product reference.
  - Managing a list of phone numbers for contacts, where you need to query or update individual numbers.
- Use multi-value types when:
  - Storing tags for a blog post, where the tags are simply a list of strings with no additional attributes.
  - …

---

14 2NF

The table is in 2NF iff :

- The table is in 1NF,
- it has a single attribute unique identifier (UID),
- and every non key attribute is dependent on the entire UID

Some cases of non-compliance with 2NF

- Derived or calculated fields:
  - Relation: Employee(EmployeeID, Name, BirthDate, Age)
  - Here, Age is derived from BirthDate, causing a partial dependency.
- Redundant information:
  - Relation: Order(OrderID, ProductID, ProductName, ProductCategory)
  - ProductCategory and ProductName both depend on ProductID, not the full OrderID.
- Inverted one to many
  - A has many Bs
  - Relation: B(Bid, B.attribute, A.attribute)
- Composite keys with partial dependencies:
  - Relation: R(A, B, C, D) where (A, B) is the composite primary key
  - If C depends only on A, we have a 2NF violation.

The 1 table version of the account phone table was not in 2NF.
see :

- Example of 2NF
- Satisfying 2NF

14.1 2NF violation in the trees table

The tree table has many 2NF violations, since all the categorical columns are into a one to many (category has many trees) relation.

2NF normalization tells us we should create a table for all the categorical columns.

But that feels overkill and instead of simplifying the logical structure of the data it complexifies it without clear gain.

Keeping the categories in the tree table is a form of denormalization.

14.2 When to apply 2NF to a categorical attribute ?

You might consider normalizing a categorical attribute (column) into a separate table if:

- The category data is large or complex (e.g., includes descriptions, hierarchies).
- Categories change frequently or need to be managed separately.
- There's a need to enforce referential integrity on categories.
- The application requires complex operations or reporting on categories.

Design evolution:

- Start with categorical attributes .
- You can always normalize later.

---

15 3NF

A table is in 3NF if and only if both of the following conditions hold:

- The table is in second normal form (2NF).
- No non-prime attribute of R is transitively dependent on the primary key.

A transitive dependency occurs when a non-prime attribute (an attribute that is not part of any key) depends on another non-prime attribute, rather than depending directly on the primary key.

where: * non-prime attribute: an attribute that is not part of any key

It's becoming a bit abstract :)

15.0.1 Example

Student(StudentID, Name, CourseID, CourseName, InstructorName)

This relation violates 3NF because:

- CourseID → CourseName (CourseName depends on CourseID, not on StudentID)
- CourseID → InstructorName (InstructorName depends on CourseID, not on StudentID)

Here, we have transitive dependencies: * StudentID → CourseID → CourseName * StudentID → CourseID → InstructorName

3NF

15.1 Difference between 2NF and 3NF

3NF is very similar to 2NF

The key differences:

- Nature of the dependency:
  - 2NF addresses partial dependencies on a key.
  - 3NF addresses transitive dependencies through non-key attributes.
- Scope of the problem:
  - 2NF focuses on the relationship between non-key attributes and parts of the key.
  - 3NF looks at dependencies between non-key attributes.

()
*
0.0417# De- nor- mal- ize

()
*
0.0417When to de- nor- mal- ize ?

()
*
0.0417Dim- itri Fontaine - The Art of Post- greSQL (2022) The Art of Post- greSQL

()
*
0.0417p
283

()
*
0.0417*Once you have thoroughly normalized your database, some difficulties may appear, either because the fully normalized schema is too heavy to deal with at the application level without any benefits, or because*

()

*

0.0417*Fully normalized schemas often have a high number of tables and references in between them. That means lots of foreign key constraints and lots of join operations in all your application queries.*

()
*
0.0417 *When there's no way to speed-up your application another way, then it is time to denormalize the schema, i.e. make a decision to put your data quality at risk in order to be able to serve your users and*

()
*
0.0417In
short:
**Keep
it
sim-
ple**

16 PostgreSQL multi-valued data types

PostgreSQL offers several data types that can store multiple values in a single column, which breaks the First Normal Form (1NF).

These types are often used for performance optimization, improved query efficiency, or when the data naturally fits a more complex structure.

16.0.1 Array Types:

Allows storing multiple values of the same type in a single column.

Example: `INTEGER[]`, `TEXT[]`

When to use: * When you need to store a fixed or variable number of elements of the same type. * For implementing tags, categories, or any list-like data where order matters. * When you frequently need to query or update the entire list as a unit.

16.0.2 JSONB (and JSON):

Stores JSON data with indexing and querying capabilities

When to use: * For semi-structured data that doesn't fit well into a relational model. * When your application needs to store flexible, schema-less data. * For data that is frequently read but less frequently updated. * When you need to query or index JSON data efficiently.

also : HSTORE, RANGE …

16.0.3 Composite types

User-defined type that combines multiple fields into a single column.

Example: CREATE TYPE address AS (street TEXT, city TEXT, zip TEXT);

When to use:

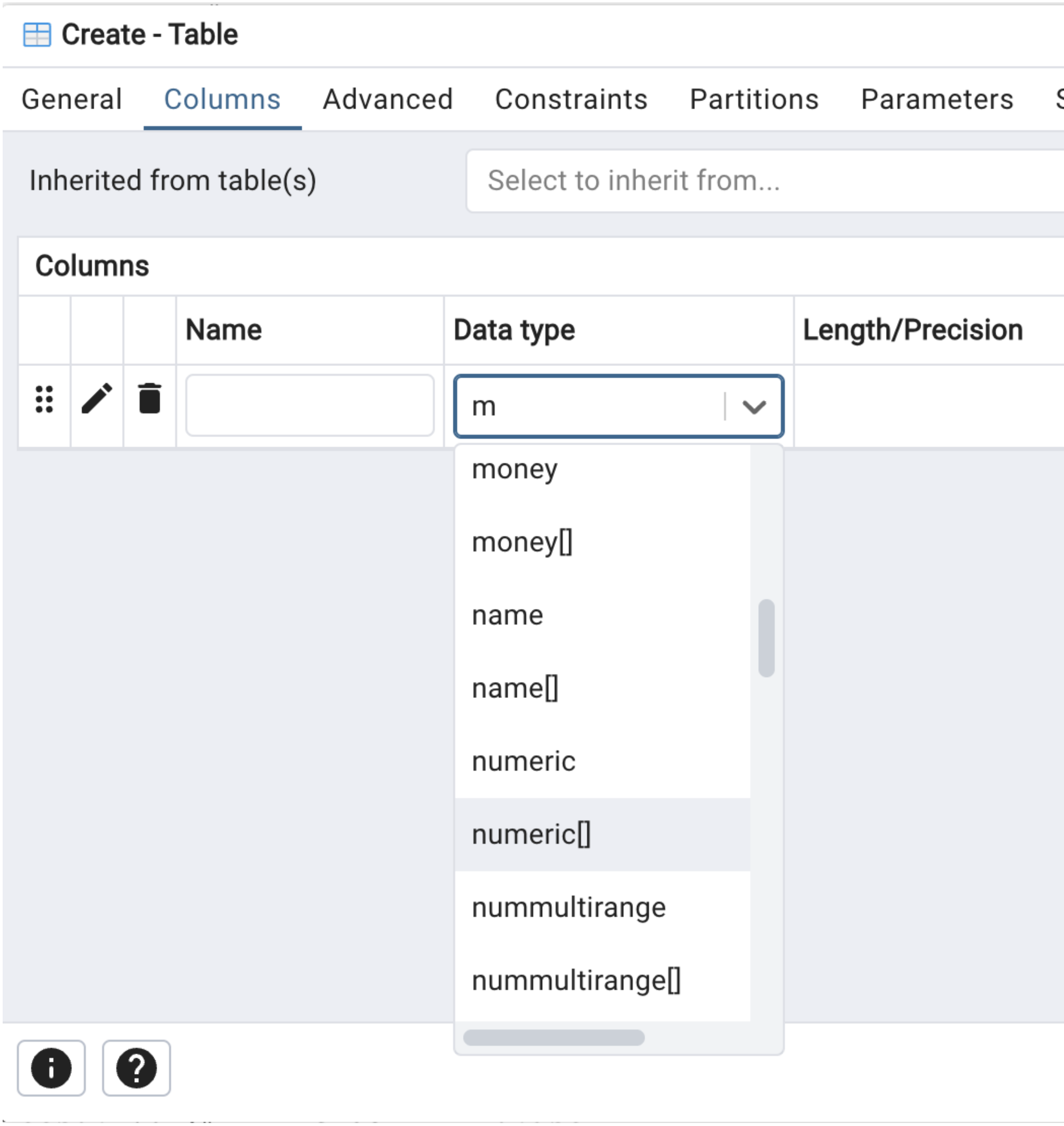
- When you have a logical grouping of fields that are always used together.
- For improving query performance by reducing joins.
- When you want to enforce a structure on a group of related fields.

17 Many available data types in PostgresQL

There are many data types in postgreSQL

see https://www.postgresql.org/docs/current/datatype.html

see also the long list of availabe data types when adding a column in pgAdmin


Create - Table
General
Columns
Advanced
Constraints
Partitions
Parameters
Inherited from table(s)
Select to inherit from...
Columns
Name
Data type
Length/Precision
m
money
money[]
name
name[]
numeric
numeric[]
nummultirange
nummultirange[]