Andrey Zavadskiy

PASS
SQLSATURDAY
ISRAEL | 26 APR 2018

Database Modeling In Practice

About me

- Solutions architect, SQL & .NET developer
- Interests: SQL Server, Entity Framework, Backend, MVC

http://andreyzavadskiy.com

https://www.facebook.com/andrey.k.zavadskiy

@AndreyZavadskiy

https://www.linkedin.com/in/zavadskiy

Huge thanks to our amazing sponsors!

Global SQLSaturday Partner

Microsoft PASS

Gold Sponsor

Quest SQREAM inspire TECHNOLOGIES EXPERDA Azure

Silver Sponsor

dbWatch DATABASE CONTROL VALINOR DATASITE

Bronze Sponsor

Prologic

Three “KNOW” to win

- Know your model
- Know your data
- Know how your data is used

Agenda

- Conceptual design
- Logical design
- Physical design

Conceptual design

Conceptual model

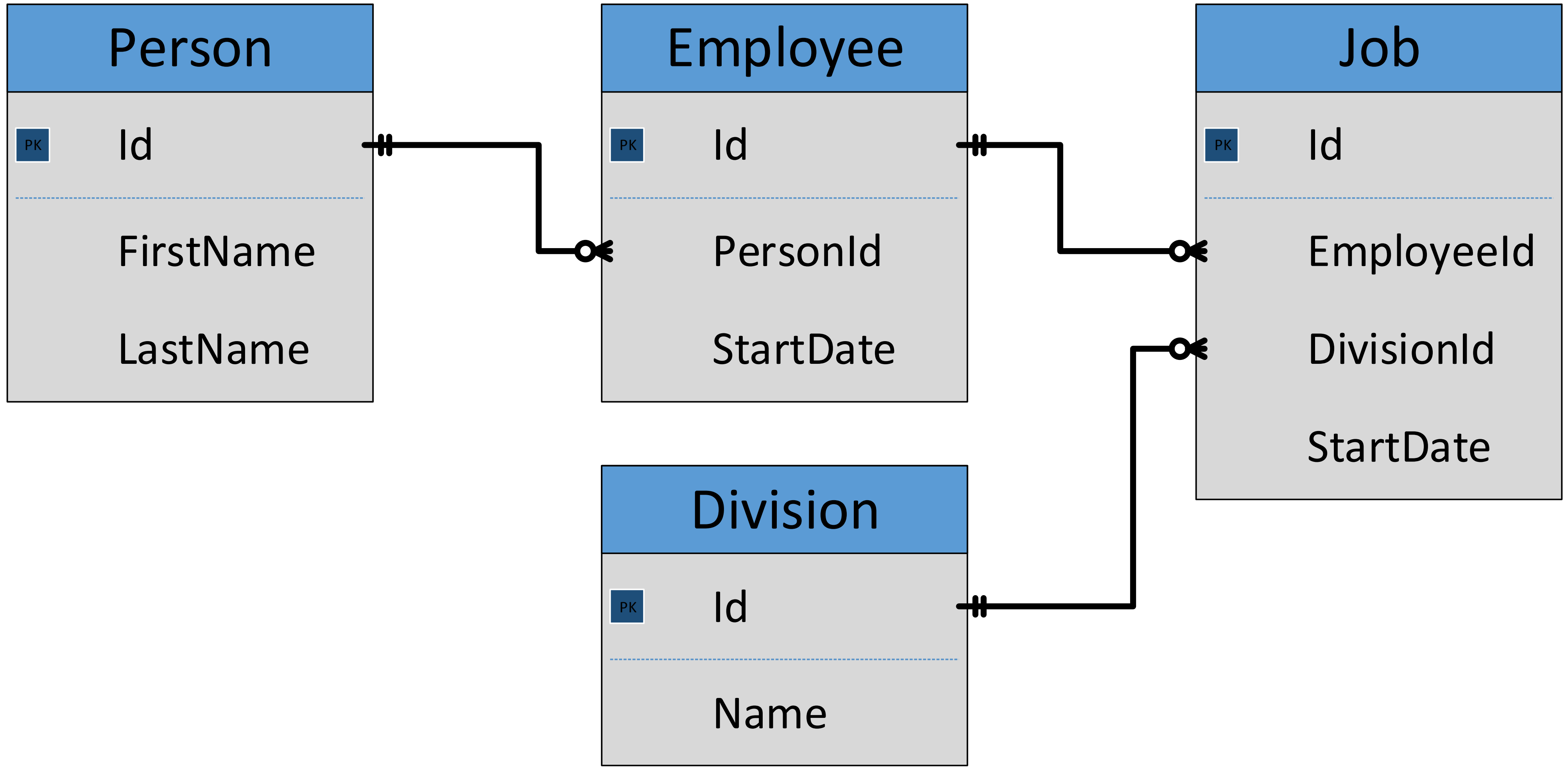
Involves:

- Entities
- Attributes
- Relationships

Derived from business objects and business requirements

Represented in Entity-Relationship (ER) diagram

ER diagram


Person
PK
Id
FirstName
LastName
Employee
PK
Id
PersonId
StartDate
Job
PK
Id
EmployeeId
DivisionId
StartDate
Division
PK
Id
Name

Entity and time scale (1)

Infinite life of entities

- Usually represents object or state
- Changes are unpredictable
- Examples: person, employee, country, currency name
- Leads to: 1 record for 1 object

Entity and time scale (2)

Time series

- Always have concrete moments of start and finish
- Usually have multiple, very often adjacent, periods of life
- Examples: jobs, sale prices, currency rates
- Need additional attributes: start/end dates
- Lead to: many records for 1 object

Changes aka History

- Do you need to save history?
- Depth of history:
  - Only last
  - Within time frame, i.e. financial year
  - All
- Amount of history tracking:
  - Events (logging)
  - Only important data
  - All attributes

Slow changing dimensions (1)

- Originally designed for data warehouses
- Include types from 0 to 6

https://en.wikipedia.org/wiki/Slowly_changing_dimension

Slow changing dimensions (2)

Type 0 – retain original

Attribute will never change

Example: birthday

Slow changing dimensions (3)

Type 1 – overwrite

Before:

Id	Supplier_Code	Supplier_Name	Supplier_State
123	ABC	Acme Supply Co	**CA**

After:

Id	Supplier_Code	Supplier_Name	Supplier_State
123	ABC	Acme Supply Co	**IL**

Slow changing dimensions (4)

Type 2 – add new row

Needs additional columns for version number and/or start-end dates

Id	Supplier_Code	Supplier_Name	Supplier_State	StartDate	EndDate
123	ABC	Acme Supply Co	**CA**	01.01.2015	14.01.2018
124	ABC	Acme Supply Co	**IL**	15.01.2018	

Slow changing dimensions (5)

Type 3 – add new attribute

Remembers only current and previous values

Id	Supplier_Code	Supplier_Name	Original_Supplier_State	Effective_Date	Current_Supplier_State
123	ABC	Acme Supply Co	**CA**	15.01.2018	**IL**

Slow changing dimensions (6)

Type 4 – add history table

Base table

Id	Supplier_Code	Supplier_Name	Supplier_State
123	ABC	Acme & Johnson Supply Co	IL

History table

Id	Supplier_Code	Supplier_Name	Supplier_State	Create_Date
1027	ABC	Acme Supply Co	CA	01.01.2015
1159	**ABC**	**Acme & Johnson Supply Co**	**IL**	15.01.2018

Change as Business Fact

- Some changes mean business facts/events
- Can be used in reporting
- Can be a basis for data warehouse

Delete

- Revert
    - Rollback erroneous operation
- Hard delete
    - Record is physically removed from table
- Soft delete
    - Record is only marked as deleted
    - Needs additional attribute

Archiving

- Confused with zip archives or backups
- Implies physical movement of data to a dedicated storage
- Application should be capable to lookup archived data
- Should support reverse operation

Logical design

Logical model

Gives detailed description for:

- Entities as tables
- Attributes as columns/fields
- Relationships as foreign keys

Usually normalized to at least 3rd normal form

Entity or Attribute dilemma

- Key question: Does attribute can have multiple values?
- Usually leads to create an entity and relationship
- Difficult situations:
  - Is lastname an entity or attribute?
  - Phone number?

One or many values

Person lastname

- Could change after marriage
- Likely to be different in national and international passport
- Dual citizenship

Additional entities

Many-to-many relationship

- Implemented via junction table

Multi-valued attribute

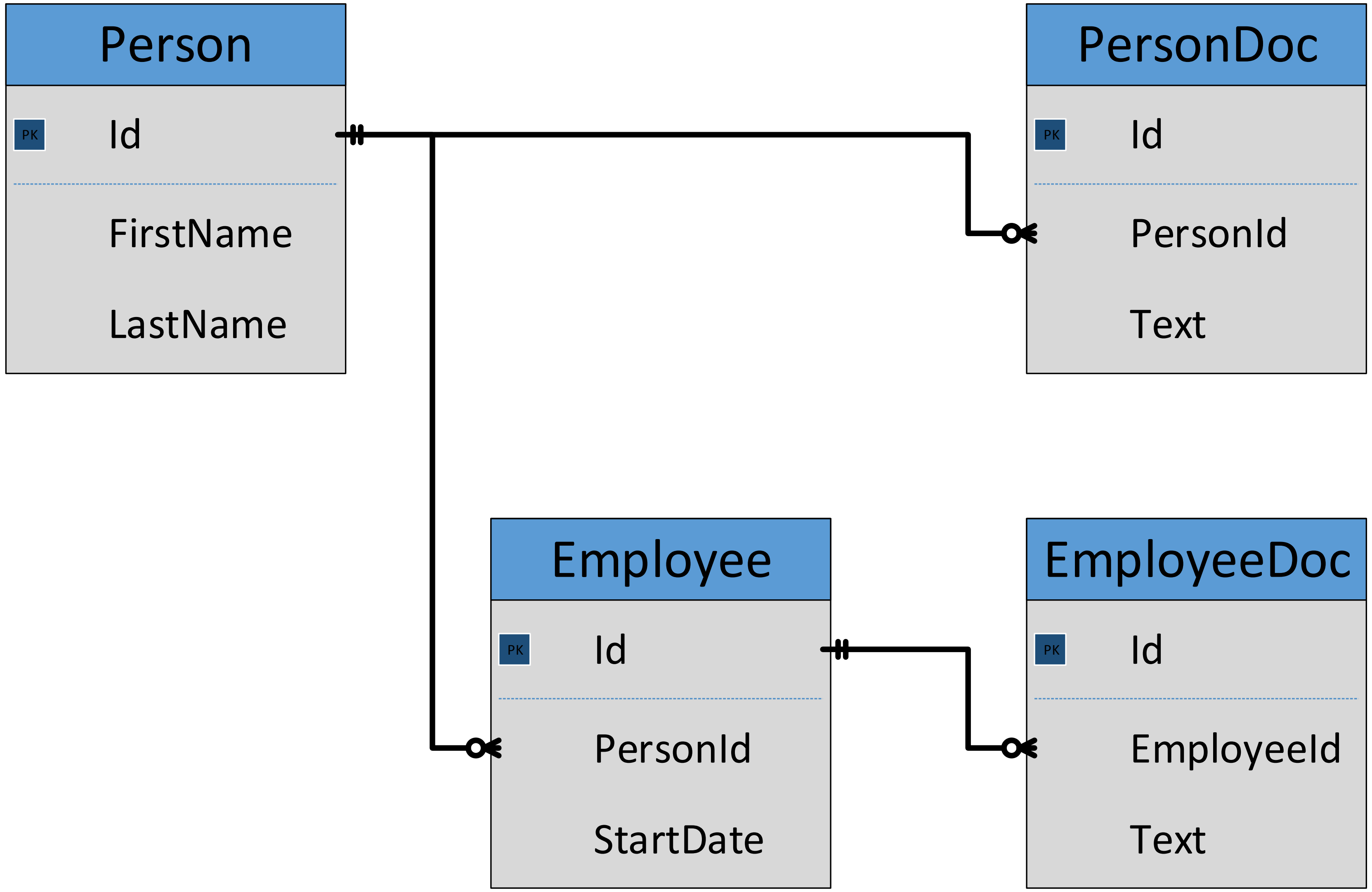
- Prefer to use separate entity (table)

Too many attributes

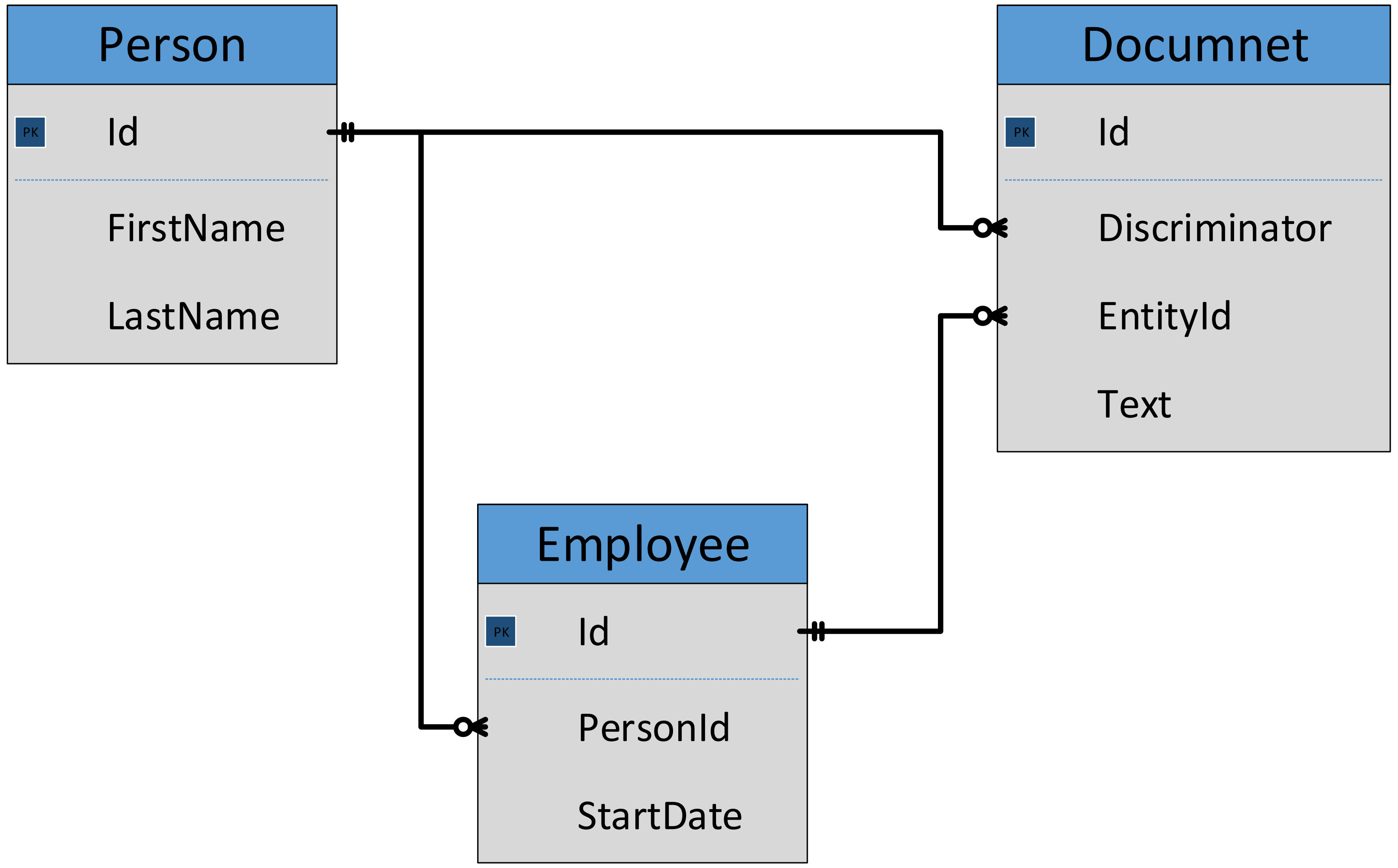
Possible implementation:

- Normalized table
- Name/Value pairs table
- Properties as XML/JSON or binary serialized structure
- SPARSE columns

Generalization (1)



Generalization (2)



Physical design

Physical model

- Table
- Columns
- Primary/foreign keys
- Constraints
- Indexes and indexed views

Primary key

- Candidates: int, guid
- Avoid string columns and composite keys
- Physical implementation
  - Clustered key (by default)
  - Unique not null index
- Key generators: identity, sequence, NEWID()

Time intervals

- Range - start/end date

Id	Job_Title	StartDate	EndDate
58	Junior developer	10.02.2016	18.06.2017
422	Mid-developer	19.06.2017	*null*

- Effective date

Id	Job_Title	EffectiveDate
58	Junior developer	10.02.2016
422	Mid-developer	19.06.2017
957	*Not working*	15.01.2018

Storage options

- Database page types
  - Data row
  - Row overflow
  - LOB
- Different filegroups
- Indexes and indexed views
- Sparse columns
- Filestream

Saving changes

- Format: string, table, XML/JSON
- Where
  - One place (suits for logging)
  - Many places (best for per table history)
- How:
  - Stored procedure, database trigger
  - Replication, Change Data Capture
  - Application

Denormalization (1)

Goal – improve performance by:

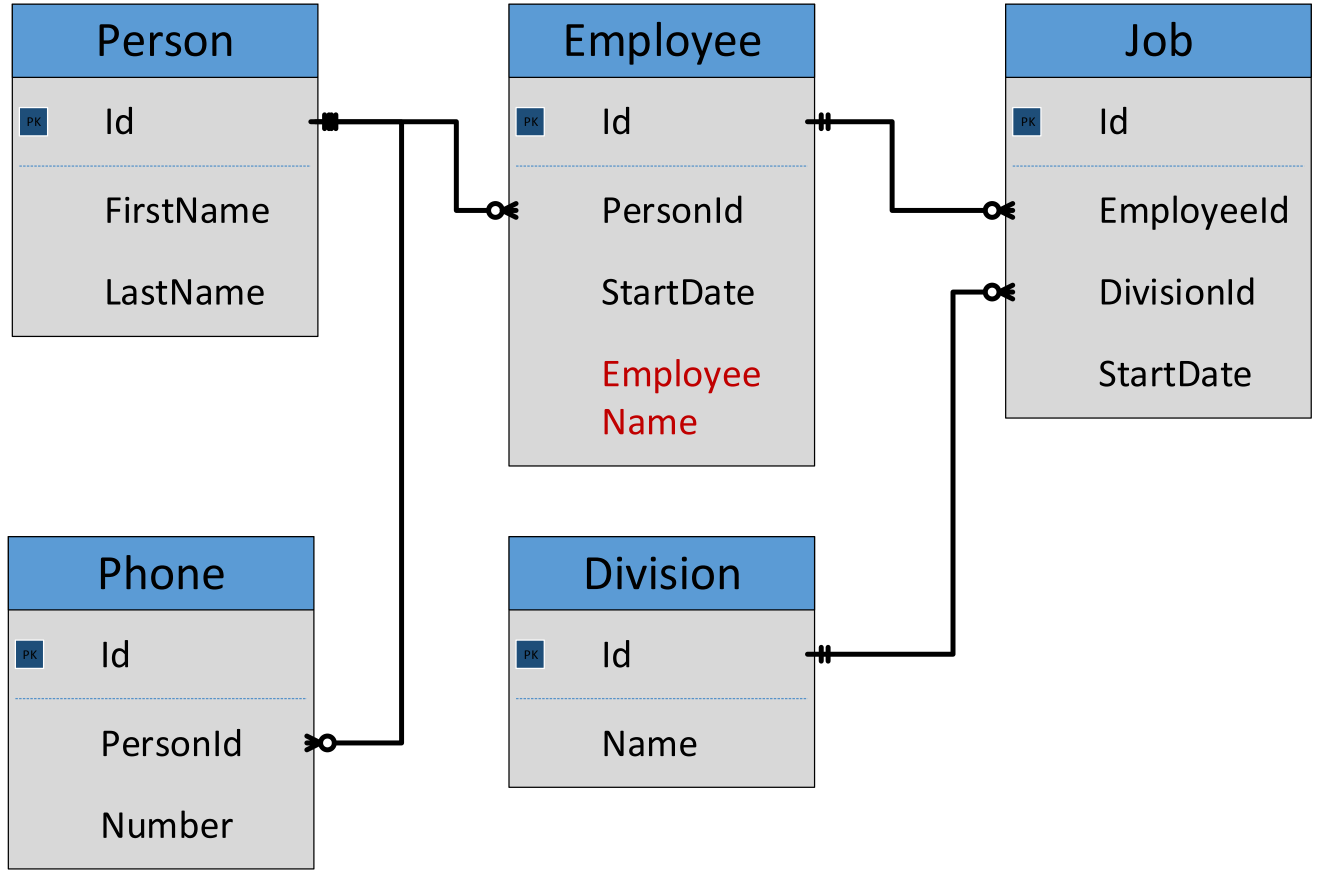
- Reducing memory operations
- Reducing CPU calculations

Minimize:

- Joins (Employee name as Person name + DivisionName + First phone number)
- Aggregates (Order total price/total weight)
- String group concatenation (all phone numbers as one string)

Denormalization (2)

Employee name: Bob Marley, Q&A, +1(555)111-2233



Denormalization (3)

Reducing memory operations

- Storage overhead in additional column for data from another table or aggregates
- Must be updated on changes (by application or trigger)

Reducing CPU calculations

- Less joins and aggregations => less CPU load
- Persisted calculated fields

Partitioning

Vertical

- Can query smaller table
- Needs join for querying all columns

Horizontal (partition table)

- Enterprise edition, Standard edition since SQL Server 2016 SP1
- Physical data movement on partition split/merge

Partition view

- Can be updatable

Questions?

Thanks!