Database Systems Lab

October 1, 2025

Inheritance and Optional Features

AGENDA

Postgres Inheritance Implementation

Gian Andrea Daniel Gerber

Primary Key - Foreign Key Implementation

Omkar Ingale | Arjun Roy

Additional Features

Juan de Vivero Woods

SQL Create Queries with inheritance

```sql
CREATE TABLE VEHICLE (
    ID SERIAL NOT NULL PRIMARY KEY,
    LICENSEPLATE VARCHAR(20),
    MAKE VARCHAR(10),
    MODEL VARCHAR(20),
    MOTORNUMBER BIGINT,
    KILOMETERS PUBLIC.KILOMETER,
    ISOPERATIONAL BOOLEAN,
    OPERATIONSTART TIMESTAMP,
    LASTSERVICE TIMESTAMP,
    GASCONSUMPTION PUBLIC.LITER,
    MAXKILOMETERS PUBLIC.KILOMETER,
    PRICEPERKILOMETER MONEY,
    PRICEPERHOUR MONEY
);
```

```sql
CREATE TABLE MOTORCAR (CHILDSEAT BOOLEAN, EXTRAS VARCHAR(30) ARRAY)
INHERITS (VEHICLE);
```

```sql
CREATE TABLE LIMOUSINE (
    PRICEPERKILOMETER MONEY DEFAULT 5,
    PRICEPERHOUR MONEY DEFAULT 100,
    MAXKILOMETERS PUBLIC.KILOMETER DEFAULT 200000,
) INHERITS (MOTORCAR);
```

Querying

```
select * from van;
```

Data Output | Messages | Notifications

Showing rows: 1 to 3 | Page No: 1 of 1

	id integer	licenseplate character varying (20)	make character varying (10)	model character varying (20)	motornumber bigint	kilon num
1	16	ZH4501	Ford	Transit	45001	
2	17	ZH4502	Mercedes	Sprinter	45002	
3	18	ZH4503	Renault	Master	45003	

Query | Query History

```
select * from vehicle;
```

Data Output | Messages | Notifications

Showing rows: 1 to 24 | Page No: 1 of 1

	id [PK] integer	licenseplate character varying (20)	make character varying (10)	model character varying (20)	motornumber bigint
1	1	ZH4001	VW	Polo	40001
2	2	ZH4002	Opel	Corsa	40002
3	3	ZH4003	Fiat	Punto	40003
4	4	ZH4101	Mercedes	S-Class	41001
5	5	ZH4102	BMW	7-Series	41002
6	6	ZH4103	Audi	A8	41003
7	7	ZH4201	BMW	Z4	42001
8	8	ZH4202	Mazda	MX-5	42002
9	9	ZH4203	Mercedes	SLK	42003
10	10	ZH4301	Toyota	Camry	43001
11	11	ZH4302	Honda	Accord	43002
12	12	ZH4303	Nissan	Altima	43003
13	13	ZH4401	Volvo	V90	44001
14	14	ZH4402	Skoda	Octavia Combi	44002

Primary Key - Foreign Key Implementation

1. Table Per Class Approach
2. Serial Primary Key instead of License Plate
3. Primary Key shared across tables

SQL Create Queries

```sql
-- Create custom domains for specific data types
CREATE DOMAIN KILOMETER AS INTEGER CHECK (VALUE >= 0);
CREATE DOMAIN LITRE AS DECIMAL(10,2) CHECK (VALUE >= 0);
CREATE DOMAIN MONEY AS DECIMAL(10,2) CHECK (VALUE >= 0);

-- Root table: Vehicle
CREATE TABLE Vehicle (
    vehicleId SERIAL PRIMARY KEY,
    licensePlate VARCHAR(20) NOT NULL UNIQUE,
    make VARCHAR(50) NOT NULL,
    model VARCHAR(50) NOT NULL,
    motorNumber INTEGER NOT NULL,
    kilometers KILOMETER NOT NULL DEFAULT 0,
    isOperational BOOLEAN NOT NULL DEFAULT TRUE,
    operationStart DATE NOT NULL,
    lastService DATE,
    gasConsumption LITRE NOT NULL,
    picture BYTEA,
    maxKilometers KILOMETER,
    pricePerKilometer MONEY,
    pricePerHour MONEY
);
```

SQL Create Queries

```sql
-- Second level: MotorCar (inherits from Vehicle)
CREATE TABLE MotorCar (
    vehicleId INTEGER PRIMARY KEY,
    childSeat BOOLEAN NOT NULL DEFAULT FALSE,
    extras VARCHAR(500),

    -- Foreign key constraint to Vehicle
    CONSTRAINT fk_motorcar_vehicle
        FOREIGN KEY (vehicleId)
        REFERENCES Vehicle(vehicleId)
        ON DELETE CASCADE
);

-- Second level: Truck (inherits from Vehicle)
CREATE TABLE Truck (
    vehicleId INTEGER PRIMARY KEY,
    maxLoad INTEGER NOT NULL CHECK (maxLoad > 0),

    -- Foreign key constraint to Vehicle
    CONSTRAINT fk_truck_vehicle
        FOREIGN KEY (vehicleId)
        REFERENCES Vehicle(vehicleId)
        ON DELETE CASCADE
);
```

SQL Create Queries

```sql
-- Third level: CompactCar (inherits from MotorCar)
CREATE TABLE CompactCar (
    vehicleId INTEGER PRIMARY KEY,

    -- Foreign key constraint to MotorCar
    CONSTRAINT fk_compactcar_motorcar
        FOREIGN KEY (vehicleId)
        REFERENCES MotorCar(vehicleId)
        ON DELETE CASCADE
);

-- Third level: Convertible (inherits from MotorCar)
CREATE TABLE Convertible (
    vehicleId INTEGER PRIMARY KEY,
    backSeat BOOLEAN NOT NULL DEFAULT TRUE,

    -- Foreign key constraint to MotorCar
    CONSTRAINT fk_convertible_motorcar
        FOREIGN KEY (vehicleId)
        REFERENCES MotorCar(vehicleId)
        ON DELETE CASCADE
);
```

SQL Inserts

```sql
-- 1. Insert into Vehicle
INSERT INTO Vehicle (
    licensePlate, make, model, motorNumber, kilometers, operationStart,
    lastService, gasConsumption, pricePerKilometer, pricePerHour
) VALUES (
    'ABC123', 'Audi', 'A5 Cabriolet', 1001, 25000, '2022-01-10',
    '2023-12-01', 7.5, 0.50, 15.00
);

-- 2. Assume vehicleId generated was 1
--    Insert into MotorCar
INSERT INTO MotorCar (
    vehicleId, childSeat, extras
) VALUES (
    1, TRUE, 'Heated seats, Bluetooth'
);

-- 3. Insert into Convertible
INSERT INTO Convertible (
    vehicleId, backSeat
) VALUES (
    1, TRUE
);
```

SQL Inserts

Query | Query History | Scratch Pad

```sql
SELECT * FROM vehicle
```

Data Output | Messages | Notifications

Showing rows: 1 to 4 | Page No: 1 of 1

	vehicleid [PK] integer	licenseplate character varying (20)	make character varying (50)	model character varying (50)	motornumber integer	kilometers integer	isoperational boolean	operationstart date	lastservice date	gasconsumpti numeric (10,2)
1	1	ABC123	Audi	A5 Cabriolet	1001	25000	true	2022-01-10	2023-12-01	7.
2	2	TRK789	Mercedes	Sprinter 316	2002	80000	true	2020-06-15	2023-11-10	12.
3	3	VAN456	Volkswagen	Multivan	3003	45000	true	2021-03-20	2024-02-28	9.
4	4	STW321	Skoda	Octavia Combi	4004	60000	true	2019-05-10	2024-05-01	6.

SQL Inheritance v/s Primary-Foreign Key Modeling:

A Comparison

SQL Inheritance

Pros:

- Code Reusabilty: Common columns defined in the parent table are inherited by child tables, less duplication.
- Simple hierarchical model: Fits naturally with object-oriented thinking (e.g., Vehicle → Car → Convertible).

Cons:

- Not standard SQL : Most databases (e.g., MySQL, Oracle, MS SQL Server) do not support INHERITS, only PostgreSQL does.
- Bad query performance:: In large parent-child hierarchies, queries on the parent table might become inefficient.

Primary Key-Foreign Key Modeling

Pros:

- Standard SQL: Works in virtually all relational databases (PostgreSQL, MySQL, Oracle, SQL Server, etc.).
- Clear integrity rules: Foreign keys ensure strong referential integrity; constraints are automatically enforced.
- Explicit relationships: Structure is transparent, making it easier to understand, debug, and extend

Cons:

- More verbose: Requires more JOINs in queries, more boilerplate code.
- Multi-step insertions: Need to insert sequentially into the base table first, then into the subtypes

.

SQL Inheritance v/s Primary-Foreign Key Modeling:

A Comparison

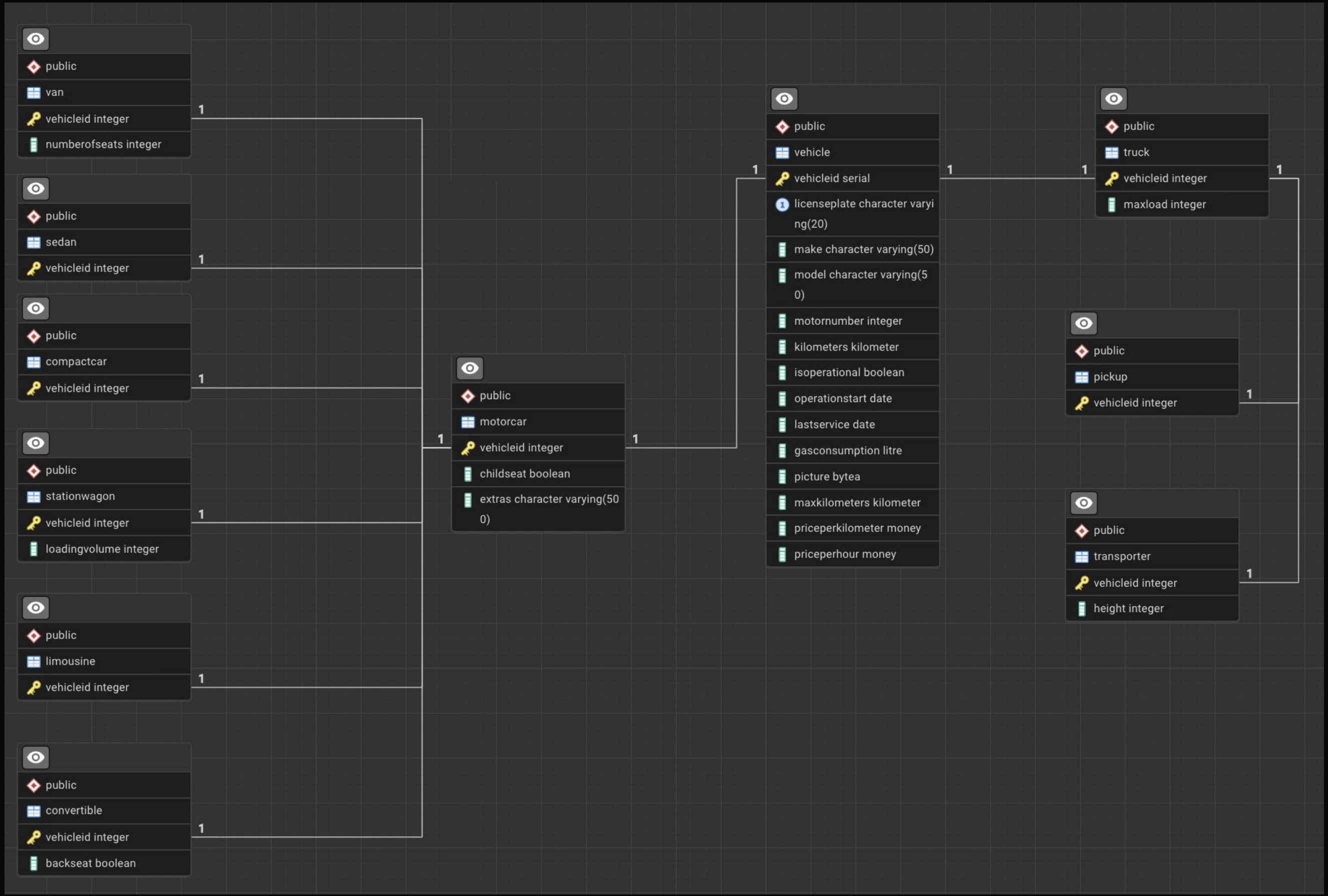
Use SQL Inheritance when:

- ❏ Desire quick prototyping in PostgreSQL.
- ❏ Favor object-oriented modeling.
- ❏ Need parent-level queries across all subtypes.

Use PK-FK Normalization when:

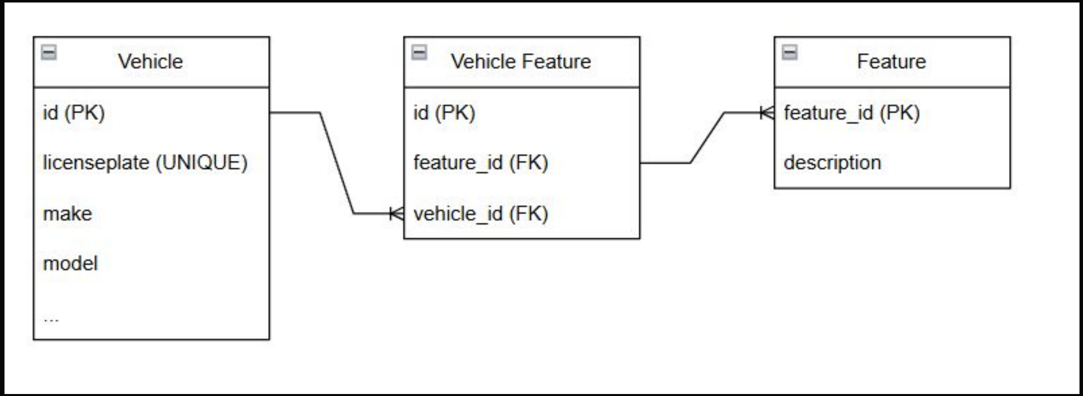
- ❏ Need portability across databases.
- ❏ Desire strong referential integrity.
- ❏ Want to clearly enforce constraints and business logic per subtype.

ERD FROM SCHEMA



Features Implementation

Approach 1 – Normalized (Feature + VehicleFeature)



-Each VEHICLE can have many Features through the VehicleFeature junction.

-The junction table holds the set-valued attribute (extras).

-Simple queries: “find all vehicles with GPS” join through VehicleFeature + Feature.

Pros

-One catalog of features → no duplication, consistent definitions.

-Many-to-many flexibility → any vehicle can have any combination of features.

-Easy cross-type queries: “show all vehicles with GPS” works in one query across all subtypes.

-Strong data integrity: FKs guarantee features are valid.

Cons

-Requires an extra junction table (VehicleFeature).

-Queries involve joins

-Slightly more overhead for inserts (must insert into two tables instead of one).

Best suited for:

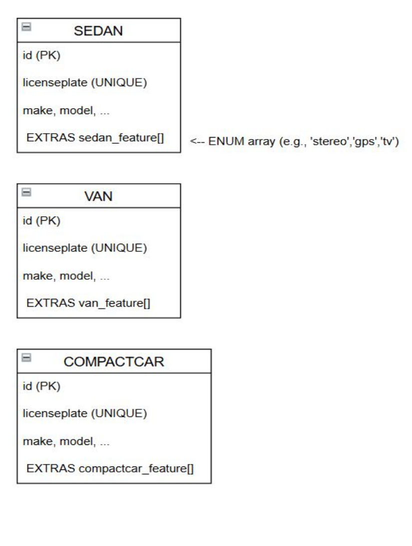
-Real production databases where reporting/analytics across all vehicles is important.

-Systems that will evolve with many feature types.

-Cases where you want to avoid inconsistent data.

Features Implementation

Approach 2 – Features stored in subtype tables (ENUM[])



- Each subtype keeps its own EXTRAS as an ENUM array. Very simple, no joins, values are validated by the enum type.

Pros

-Very simple: features live right in each table
-No joins needed: easy to query a subtype directly.
-ENUM validation: prevents typos and enforces allowed values.

Cons
-Schema changes needed to add new features (`ALTER TYPE … ADD VALUE`).
-Per-subtype duplication: “stereo” may exist in Sedan enum, Van enum, etc.
-Harder cross-type queries: you need `UNION` across subtypes to find “all vehicles with GPS.”
-Not scalable if the number of features or subtypes grows.

Best suited for:

-Small/demo systems.
-Situations where features are stable and don’t change often.
-Teaching conceptual designs, where simplicity matters more than scalability.

Thank You!