enovafinancial™

Table Inheritance

Jim Nasby, Lead Database Architect
Enova Financial

REAL inheritance - none of that silly partitioning stuff!

Overview

- What is inheritance?
- What's the alternative?
- ~~Problems~~Challenges
- Inheritance framework
- Metacode

enovafinancial™

What is it?

What is it?

Think object-oriented inheritance applied to a table.

```
CREATE TABLE parent(
  a    int    NOT NULL
  , b  int    CHECK( b > 0 )
);
CREATE TABLE child(
  c    int
) INHERITS( parent );
```

What is it?

```
\d child

 Column  |   Type    | Modifiers
---------+-----------+-----------
 a       | integer   | not null
 b       | integer   |
 c       | integer   |
Check constraints:
  "parent_b_check" CHECK (b > 0)
Inherits: parent
```

Data

Inheritance deals with data as well as table structure

```
INSERT INTO child VALUES( 1, 2, 3 );
SELECT * FROM parent;
```

```
 a | b
---+---
 1 | 2
(1 row)
```

enovafinancial™

Data

SELECT * FROM ONLY parent;

```
 a | b
---+---
(0 rows)
```

Data

INSERT INTO parent VALUES(8, 9);

```
 a | b
---+---
 8 | 9
 1 | 2
(2 rows)
```

Data

SELECT * FROM child;

```
 a | b
---+---
 1 | 2
(1 row)
```

enovafinancial™

Real World:

Storing customer account information

enovafinancial™

Customer Accounts

Customers have different accounts for sending and receiving money:

- Bank
- Debit / Credit card
- Paycard

enovafinancial™

Customer Accounts

Some attributes are common across all these different types of accounts

- account_id
- customer_id
- account_status_id

Customer Accounts

Some attributes are unique to specific types of accounts

- routing_number / account_number
- card_token

Your database doesn't have inheritance?

Customer Accounts

How can you reconcile the different fields?

- One table, lots of null fields (single-table inheritance)
  - customer_id NOT NULL
  - routing_number NULL
  - account_number NULL
  - card_token NULL

Customer Accounts

Master table for common stuff referenced by other tables for detailed stuff

```
SELECT …
  FROM customer_account c
    JOIN bank_account b
USING( account_id )
```

Use inheritance!

Inheritance gives the best of both worlds!

```
CREATE TABLE customer_account(...);

CREATE TABLE bank_account(...)
   INHERITS( customer_account );

CREATE TABLE debit_card(...)
   INHERITS( customer_account );
```

Inheritance

- customer_account has common fields
- bank_account and debit_card have common **and** specific fields
- Data is only stored once (in the child table)
- Which table you select from depends on what you're trying to do... is it something generic or is it specific?
  - List of customer accounts
  - Send money to customer

Problems

^$&#(@%*!

I know it was too easy!

Problems

You can't use a parent table as the target of a foreign key

- This works:

ALTER TABLE child ADD FOREIGN KEY

- So does this:

ALTER TABLE foo FOREIGN KEY child

- This does not:

ALTER TABLE foo FOREIGN KEY parent

Problems

You can't create a cross-table unique constraint (so you can't create a PRIMARY KEY)

Problems

There's a lot of things that don't carry over from a parent to a child

- Check and not null constraints carry
- Foreign key constraints do not
- Triggers do not

Also, sometimes you don't want a constraint to carry to all children

Primary / Foreign Keys

Primary / Foreign Keys

Luckily, it's ~~easy~~possible to re-create the foreign key infrastructure using user-created triggers:

- Trigger on referenced table to deal with updates and deletes
- Trigger on referring table to take a share lock on referenced record

Primary / Foreign Keys

Unfortunately, inheritance makes this more difficult...

- You can't easily create a trigger on a parent and all it's children (though, we created a framework that makes that easier)
- You can't SHARE LOCK via the parent table

ERROR: SELECT FOR UPDATE/SHARE is not supported for inheritance queries

Primary / Foreign Keys

There are two ways to deal with the share lock issue:

- Brute-force: try locking the needed row in every child table
- Smart: have a method of knowing exactly what table a child row exists in

Primary / Foreign Keys

All of our parent tables have some kind of "type" field that identifies what child table each record belongs to:

- payment_instrument_type_id
- transfer_method_id

Primary / Foreign Keys

We use that type field to find the appropriate child table:

```
SELECT INTO v_table_name table_name
  FROM payment_instrument_types
  WHERE payment_instrument_type_id =
   ( SELECT payment_instrument_type_id
        FROM payment_instruments
        WHERE payment_instrument_id =
          NEW.payment_instrument_id )
```

Primary / Foreign Keys

And then we grab a share lock:

```
EXECUTE 'SELECT 1 FROM ' ||
   v_table_name || ' WHERE id = ' ||
   NEW.payment_instrument_id || ' FOR
   SHARE';
```

Primary / Foreign Keys

Primary keys are a serious problem:

- In a btree unique index, the index insertion code locks an internal structure to ensure that no duplicate entries can be created
- The only way we could emulate that in userspace would be to have a single master table that had all the IDs and a unique index... but that's a lot of overhead

Primary / Foreign Keys

Instead, we have a trigger that fakes a unique constraint:

- Disallow deletes
- Disallow changing the ID field
- On insert assert WHERE NOT EXISTS()
- All ID fields are driven by a single sequence

There are race conditions a mile-wide in this scheme!

enovafinancial™

Clear as mud?

enovafinancial™

Inheritance Framework

Inheritance Framework

Who wants to manually add things every time you create a new child table?

- I hate typing
- I'm lazy
- I'm forgetful
- I have better things to do with my time (beer)

enovafinancial™

Inheritance Framework

Our "inheritance framework" makes it easy to add things to child tables:

- Constraints
- Triggers
- Column settings (default, storage, statistics)

The framework also allows you to exclude certain child tables

Inheritance Framework

The framework

- Identifies parent and child table relationships
- Decides if an object should not be applied to a specific child
- Uses a template and provided data to create a SQL statement for each child table
- Creates the required object if it doesn't exist

enovafinancial™

Inheritance Framework: Tags and Templates

enovafinancial™

(Tag, you're it!)

enovafinancial™

Inheritance Framework: Tags

The framework supports the following tags:

- %parent_schema%
  Schema parent table is in
- %parent_name%
  Name of parent table
- %parent_full%
  Full name (schema.name) of parent
- %child_schema%
- %child_name%
- %child_full%

Inheritance Framework: Templates

The templates are stored in inheritance.object_types:

```
SELECT object_type, template, reset_template FROM inheritance.object_types;
-[ RECORD 1 ]--+-------------------------------------------------------------------------------------
object_type    | trigger
template       | CREATE TRIGGER %object_name_template% %object_body_template%
reset_template | DROP TRIGGER %object_name_template% ON %child_full%
-[ RECORD 2 ]--+-------------------------------------------------------------------------------------
object_type    | constraint
template       | ALTER TABLE %child_full% ADD CONSTRAINT %object_name_template% %object_body_template%
reset_template | ALTER TABLE %child_full% DROP CONSTRAINT %object_name_template%
-[ RECORD 3 ]--+-------------------------------------------------------------------------------------
object_type    | column set
template       | ALTER TABLE %child_full% ALTER %object_name_template% SET %object_body_template%
reset_template | ALTER TABLE %child_full% ALTER %object_name_template% DROP %object_body_template%
```

enovafinancial™

Inheritance Framework: In action!

Inheritance Framework

Each object has a name tag and a body tag:

```
ALTER TABLE %child_full%
  ADD CONSTRAINT %object_name_template%
  %object_body_template%

SELECT inheritance.child_object__add(
  name of parent table
  , type of object (from object_types table)
  , object_name_template
  , object_body_template
);
```

Inheritance Framework

Add a primary key:

```
SELECT inheritance.child_object__add(
  -- Name of parent
  'payment_instruments.payment_instruments'
  -- Type of object
  , 'constraint'
  -- object name template
  , '%child_name%__pk_payment_instrument_id'
  -- object body template
  , 'PRIMARY KEY( payment_instrument_id )'
);
```

enovafinancial™

Inheritance Framework

When you call this:

```
SELECT inheritance.child_object__add(
 'payment_instruments.payment_instruments'
   , 'constraint'
   , '%child_name%__pk_payment_instrument_id'
   , 'PRIMARY KEY( payment_instrument_id )'
);
```

You get this:

```
\d payment_instruments.debit_card
    "debit_card__pk_payment_instrument_id"
  PRIMARY KEY, btree (payment_instrument_id)
```

enovafinancial™

Inheritance Framework

Trigger:

```
SELECT inheritance.child_object__add(
  'payment_instruments.payment_instruments'
  , 'trigger'
  , '%child_name%__dupe_id'
  , $$BEFORE INSERT OR UPDATE ON %child_full%
  FOR EACH ROW EXECUTE PROCEDURE
  tg_payment_instruments_unique()$$
);

debit_cards__dupe_id BEFORE INSERT OR UPDATE
  ON payment_instruments.debit_cards
  FOR EACH ROW EXECUTE PROCEDURE
  tg_payment_instruments_unique()
```

Inheritance Framework

Column SETing (and an excluded table):

```
SELECT inheritance.child_object__add(
  'payment_instrument.payment_instrument'
  , 'column set'
  , 'customer_id'
  , 'NOT NULL'
  -- Table(s) to EXCLUDE from this object (may be an
  array)
  , 'payment_instrument.bank_account'
);

\d payment_instrument.bank_account
 customer_id    | integer    |
\d payment_instrument.debit_card
 customer_id    | integer    | not null
```

API: Add objects to inheritance.child_objects

```
\df inheritance.
child_object__add(parent, type, name, body
  [, excluded])

child_object__add_excluded_tables(child_object_
  id, excluded)
child_object__add_excluded_tables(parent, type,
  name, excluded)

child_object__remove_excluded_tables(child_obje
  ct_id, excluded)
child_object__remove_excluded_tables(parent,
  type, name, excluded)
```

API: Process tables

Process a single child:

```
process_child(child_table)
```

Process all children of a parent:

```
process_children(parent_table)
  Returns setof child table names
```

Process everything (beware of locking!):

```
process_all()
  Returns setof child table names
```

API: Reset an object

Used when you want to remove an inheritance objects from a child or children. Does NOT remove anything from inheritance.objects, so a subsequent `process*()` call will put everything back.

```
reset_child(child_table, object_type)

reset_children(parent_table, object_type)
  returns setof child table names
```

Inheritance Framework

Reset example:

Change a trigger from being BEFORE to AFTER

```
SELECT * FROM
  reset_children('parent', 'trigger');

UPDATE inheritance.child_objects
  SET object_body_template = regexp_replace(
    object_body_template, 'BEFORE', 'AFTER' )
  WHERE …

SELECT * FROM
  process_children('parent');
```

enovafinancial™

Whew!

Questions?

enovafinancial™

If you act now, we'll throw in METACODE, a $19.95 value, for ABSOLUTELY NOTHING!

Metacode

It would be difficult to write completely generic triggers to support our fake unique constraints and foreign keys.

It's easier to create a generic TEMPLATE for those triggers, and use TAG replacement to create trigger functions that are tailored for each specific inheritance structure.

(If you were able to stay awake in the last section, this should all sound very familiar ;P)

enovafinancial™

Metacode

code.inheritance_unique(

schema

Schema that the parent (inherited) table is in. The trigger funciton will be created in this schema as well

, parent_table

Parent table of inheritance chain

, primary_key

Primary key field of the parent table (only single field keys for now)

);

Metacode

code.inheritance_unique creates a trigger function that checks to see if NEW.{primary_key} already exists in the parent table

Note that there's race conditions a mile wide here, but as long as you're using a **SINGLE** sequence to drive your primary key across **all** children (and the parent), you should be OK.

Metacode

Usage:

```
code.inheritance_unique( 'payment_instruments',
   'payment_instruments', 'payment_instrument_id' );

inheritance.child_object__add(
 'payment_instruments.payment_instruments'
  , 'trigger', '%child_name%__dupe_id'
  , $$BEFORE INSERT OR UPDATE ON %child_full%
   FOR EACH ROW EXECUTE PROCEDURE
   tg_payment_instruments_unique()
  $$
);;
```

Metacode

```
code.inheritance_ri(
  schema
  , parent_table
  , primary_key
  , type_plural
  , type_singular
  , table_name_field
  , child_name_prefix
  , child_name_suffix
);
```

Metacode

schema, parent_table and primary_key are what you'd expect

type_plural and type_singular refer to the table that contains “type” information (thank Rails for the plural/singular sillyness)

table_name_field is the field in the type table that gives you the name of the table that contains records of that type

child_name_prefix and child_name_suffix are added to the data table_name_field. Default to '%schema%' and 's'.

Metacode

Example:

```
code.lookup_table_static(
  'payment_instruments', 'payment_instrument_types',
    'payment_instrument_type'
  , ', child_table text'
);
Table "payment_instruments.payment_instrument_types"
        Column           |   Type    | Modifiers
-------------------------+-----------+-----------
 id                      | smallint  | not null
 payment_instrument_type | text      | not null
 description             | text      |
 child_table             | text      |
```

Metacode

```
INSERT INTO payment_instrument_types
   VALUES
  ( 1, 'bank_account', 'bank_account' )
  , ( 2, 'debit_card', 'debit_card' )
CREATE TABLE payment_instruments.
payment_instruments(
  payment_instrument_id int
    NOT NULL DEFAULT
    nextval('payment_instrument_id_seq'
  , payment_instrument_type_id smallint
);
```

Metacode

Fortunately, there's actually some documentation...

```
SELECT * FROM code.functions;
```

enovafinancial™

Code will (hopefully) be available on pgFoundry; search for enova-tools.

Use the forum or mailing list

enovafinancial™

OK, OK! I'll shut up now!

Questions?

Want to get paid to work with this cool stuff? Send me your resume!

jnasby@enovafinancial.com