Use Case Database Usage

Contents

- What is required
- Results analysis
- Sandbox creation

What is required

Design alternatives

Given a conceptual schema

- Implementation 1
  - Create the corresponding tables
  - Add the integrity constraints
  - Execute the queries written according to this implementation
- Implementation 2
  - Create the corresponding tables
  - Add the integrity constraints
  - Execute the queries written according to this implementation

…

- Implementation n
  - Create the corresponding tables
  - Add the integrity constraints
  - Execute the queries written according to this implementation

Table creation

```
CREATE TABLE <newTable> AS (SELECT * FROM <existingTable>);
```

Integrity constraints declaration

```
ALTER TABLE <newTable>
        ADD PRIMARY KEY (<attributeName>);

ALTER TABLE <newTableRefering>
        ADD FOREIGN KEY (<attributeNameRefering>)
                REFERENCES <newTableRefered> (<attributeNameRefered>);

ALTER TABLE <newTable>
        ADD UNIQUE (<attributeName>);

ALTER TABLE <newTable>
        ALTER COLUMN (<attributeName>)
                SET NOT NULL;
```

Results analysis

Translation diagram

```
SELECT a.model, t.year, dailyutilization
FROM AircraftUtilization f
JOIN AircraftsDimension a ON a.id=f.aircraftid
JOIN TemporalDimension t ON t.id=f.timeid
GROUP BY a.model, t.year
ORDER BY a.model, t.year;
```

Declarative

Query plan

```
GroupAggregate (actual time=0.713..0.757 rows=12 loops=1)
  Group Key: a.manufacturer, t.month
  ->  Sort (actual time=0.700..0.707 rows=67 loops=1)
      Sort Key: a.manufacturer, t.month
      Sort Method: quicksort  Memory: 30kB
      ->  Nested Loop (actual time=0.200..0.557 rows=67 loops=1)
          ->  Nested Loop (actual time=0.153..0.362 rows=67 loops=1)
              ->  Hash Join (actual time=0.082..0.148 rows=67 loops=1)
                  Hash Cond: (f.employeeid = e.id)
                  ->  Seq Scan on logbookreporting f (actual time=0.018..0.036 rows=108 loops=1)
                  ->  Hash (actual time=0.051..0.051 rows=78 loops=1)
                      Buckets: 1024  Batches: 1  Memory Usage: 11kB
                      ->  Seq Scan on employeesdimension e (actual time=0.015..0.037 rows=78 loops=1)
                          Filter: (class = 'PIREP'::bpchar)
                          Rows Removed by Filter: 59
              ->  Index Scan using aircraftsdimension_pkey on aircraftsdimension a (actual time=0.002..0.002 rows=1 loops=67)
                  Index Cond: (id = f.aircraftid)
          ->  Index Scan using temporaldimension_pkey on temporaldimension t (actual time=0.002..0.002 rows=1 loops=67)
              Index Cond: (id = f.timeid)
```

Procedural

Explain plan in DBeaver

```
SELECT a.model, t.YEAR,
  sum(f.flightcycles)/(current_date-min(t.id)-SUM(f.scheduledOutOfService)-SUM(f.unscheduledOutOfService)) AS dailycycles,
  sum(f.flighthours)/(current_date-min(t.id)-SUM(f.scheduledOutOfService)-SUM(f.unscheduledOutOfService)) AS dailyutilization
FROM AircraftUtilization f
  JOIN AircraftsDimension a ON a.id=f.aircraftid
  JOIN TemporalDimension t ON t.id=f.timeid
GROUP BY a.model, t.YEAR
ORDER BY a.model, t.YEAR;
```

Execution plan - 1

Simple

Node Type	Entity	Cost	Rows	Time	Condition
Sort		8.59 - 8.60	1	0.295	
Aggregate		8.53 - 8.58	1	0.261	
Hash Join		2.77 - 7.96	33	0.195	(f.aircraftid = a.id)
Hash Join		1.74 - 6.84	33	0.120	(t.id = f.timeid)
Seq Sc	tem...	0.00 - 4.01	201	0.038	
Hash		1.33 - 1.33	33	0.032	
Seq	aircr...	0.00 - 1.33	33	0.019	
Hash		1.01 - 1.01	1	0.045	
Seq Sc	aircr...	0.00 - 1.01	1	0.035	

Explain plan in PostgreSQL

```
EXPLAIN (ANALYZE TRUE, COSTS FALSE, SUMMARY TRUE) SELECT …;
```

- ANALYZE: Executes the query and shows actual run times
- COSTS: Includes the total cost of each operation
- SUMMARY: Includes total timing information at the end
- VERBOSE: Displays additional information
  - Like the columns generated by each operation

https://www.postgresql.org/docs/current/sql-explain.html

Evaluation of performance

	QUERY PLAN
1	GroupAggregate (actual time=0.655..0.692 rows=12 loops=1)
2	Group Key: a.manufacturer, t.month
3	-> Sort (actual time=0.643..0.650 rows=67 loops=1)
4	Sort Key: a.manufacturer, t.month
5	Sort Method: quicksort Memory: 30kB
6	-> Nested Loop (actual time=0.199..0.506 rows=67 loops=1)
7	-> Nested Loop (actual time=0.167..0.348 rows=67 loops=1)
8	-> Hash Join (actual time=0.092..0.154 rows=67 loops=1)
9	Hash Cond: (f.employeeid = e.id)
10	-> Seq Scan on logbookreporting f (actual time=0.020..0.034 rows=108 loops=1)
11	-> Hash (actual time=0.052..0.052 rows=78 loops=1)
12	Buckets: 1024 Batches: 1 Memory Usage: 11kB
13	-> Seq Scan on employeesdimension e (actual time=0.015..0.035 rows=78 loops=1)
14	Filter: (class = 'PIREP'::bpchar)
15	Rows Removed by Filter: 59
16	-> Index Scan using aircraftsdimension_pkey on aircraftsdimension a (actual time=0.002..0.002 rows=1 loops=67)
17	Index Cond: (id = f.aircraftid)
18	-> Index Scan using temporaldimension_pkey on temporaldimension t (actual time=0.002..0.002 rows=1 loops=67)
19	Index Cond: (id = f.timeid)
20	Planning Time: 5.349 ms
21	Execution Time: 0.799 ms

Execution time factors

- ✓ Amount of data
- ✓ Complexity of the query
- ✓ DBMS configuration
- ? Concurrent users
  - Caching
  - Locking
- ✗ Other services or tasks in the same server
- ✗ Network distance, bandwidth and usage

Run <u>the whole script</u> three times!!!

Do not run one statement at a time!!!

We cannot control all these factors!!!

Sandbox creation

Problem

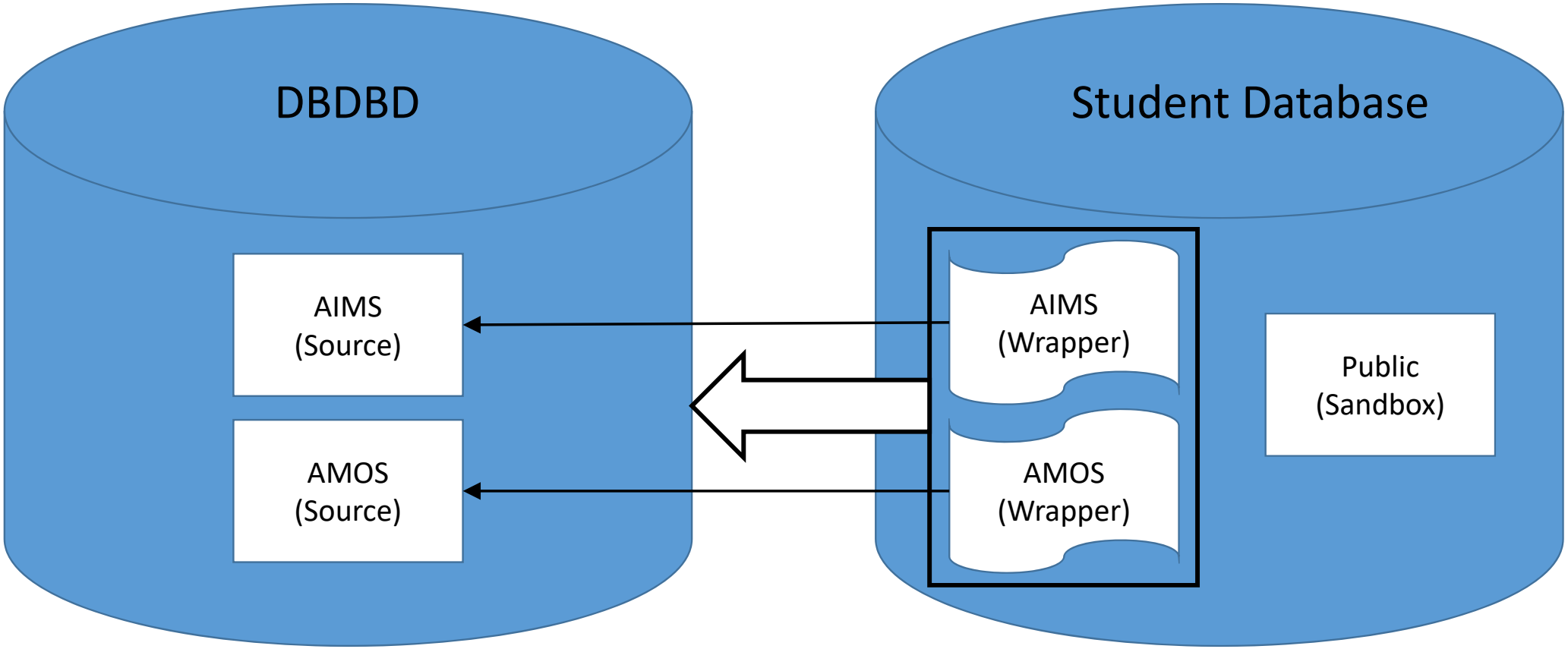
- We want to have a sandbox that can be easily reset for every session
  a) A read-only reference database
  b) A simple mechanisms to recreate the reference database in ours
     - Needs to access two databases in the same SQL sentence

```
CREATE TABLE <localTable> AS (SELECT * FROM <remoteTable>);
```

Local execution — Remote execution

Requires access the remote table from the local session!!!

Connections diagram



Correspondences between schemas

Source

- DBDBD
  - Schemas
    - AIMS
      - Tables
        - flights 20M
        - maintenance 6.1M
        - slots 8K
      - Views
      - Materialized Views
      - Indexes
      - Functions
      - Sequences
      - Data types
      - Aggregate functions
    - AMOS
      - Tables
        - aircrafts 80K
        - attachments 9M
        - forecastedorders 7.6M
        - maintenanceevents 10M
        - operationinterruption 1.8M
        - postflightreports 43M
        - technicallogbookorders 7.3M
        - workorders 8K
        - workpackages 2.5M
      - Views
      - Materialized Views
      - Indexes
      - Functions
      - Sequences
      - Data types
      - Aggregate functions

Source

Wrapper

- BDestudiantTest
  - AIMS
    - Tables
      - flights
      - maintenance
      - slots
    - Views
    - Indexes
    - Procedures
    - Sequences
    - Table Triggers
    - Data Types
  - AMOS
    - Tables
      - aircrafts
      - attachments
      - forecastedorders
      - maintenanceevents
      - operationinterruption
      - postflightreports
      - technicallogbookorders
      - workorders
      - workpackages
    - Views
    - Indexes
    - Procedures
    - Sequences
    - Table Triggers
    - Data Types
  - information_schema
  - pg_catalog
  - public

Wrapper

Sandbox

PostgreSQL mechanism: “postgres_fdw”

- Not standard
  - Extension
    - Needs to be installed by the server administrator
- Provides transparent access to remote tables

```
SELECT * FROM pg_catalog.pg_extension; -- This should show a row with “postgres_fwd”
```

https://www.postgresql.org/docs/current/postgres-fdw.html

PostgreSQL sentences template

1. Create a virtual local server connected to the remote one

```
CREATE SERVER <local_servername> FOREIGN DATA WRAPPER postgres_fdw
                OPTIONS (host '<hostname>', dbname '<database>', port '<port>');
```

2. Map the local user to the user in the remote server (in our case, we will use the same user)

```
CREATE USER MAPPING FOR <local_username> SERVER <local_servername>
                OPTIONS (user '<remote_username>', password '<remote_password>');
```

3. Create a local schema (in our case, we will use the same schema name, but it could be different)

```
CREATE SCHEMA <local_schema> [AUTHORIZATION <local_username>];
```

4. Map the local schema to the remote one

```
IMPORT FOREIGN SCHEMA <remote_schema> FROM SERVER <local_servername> INTO <local_schema>;
```

https://www.postgresql.org/docs/current/postgres-fdw.html

PostgreSQL sentences instantiation

```
CREATE SERVER myserver FOREIGN DATA WRAPPER postgres_fdw
       OPTIONS (host 'postgresfib.fib.upc.edu', dbname 'DBDBD', port '6433');

SELECT * FROM pg_catalog.pg_foreign_server; -- This should show a row with “myserver”

CREATE USER MAPPING FOR "estudiantTest" SERVER myserver
       OPTIONS (user 'estudiantTest', password 'XXXXXXXXXXXX');

CREATE SCHEMA "AIMS" AUTHORIZATION "estudiantTest";
CREATE SCHEMA "AMOS" AUTHORIZATION "estudiantTest";

IMPORT FOREIGN SCHEMA "AIMS" FROM SERVER myserver INTO "AIMS";
IMPORT FOREIGN SCHEMA "AMOS" FROM SERVER myserver INTO "AMOS";
```

The username and password are those of the DBMS (not *Raco*)!!!

Tips

- The username and password are those of the database (not the Raco)
- In Dbeaver, refresh the folders for the schemas and tables to appear
- Depending on the Dbeaver version, the remote tables appear as regular tables or in a dedicated folder
- If you make a mistake in the server, you can drop it with

```
DROP SERVER <local_servername>;
```

- If you make a mistake in the username and password, you can drop it with

```
DROP USER MAPPING FOR <local_username> SERVER <local_servername>;
```

Execute all together

```
--=================================================================================================
--=================================== Implementation with an FK in postflightreports ============================
DROP TABLE IF EXISTS technicallogbookorders CASCADE;
DROP TABLE IF EXISTS postflightreports CASCADE;

CREATE TABLE technicallogbookorders AS (SELECT * FROM "AMOS".technicallogbookorders tlb);
CREATE TABLE postflightreports AS ( SELECT * FROM "AMOS".postflightreports p);

ALTER TABLE technicallogbookorders ADD PRIMARY KEY (workorderid);
ALTER TABLE postflightreports ADD PRIMARY KEY (pfrid);
ALTER TABLE postflightreports ADD FOREIGN KEY (tlborder) REFERENCES technicallogbookorders(workorderid);

-- Title: Retrieve all technicallogbookorders with postflightreports
EXPLAIN (ANALYZE TRUE, COSTS FALSE, SUMMARY true) SELECT * FROM technicallogbookorders tlb WHERE EXISTS (
  SELECT *
  FROM postflightreports pfr
  WHERE pfr.tlborder=tlb.workorderid);

-- Title: Retrieve all technicallogbookorders without postflightreports
EXPLAIN (ANALYZE TRUE, COSTS FALSE, SUMMARY true) SELECT * FROM technicallogbookorders tlb WHERE NOT EXISTS (
  SELECT *
  FROM postflightreports pfr
  WHERE pfr.tlborder=tlb.workorderid);

-- Title: Retrieve all postflightreports with technicallogbookorders
EXPLAIN (ANALYZE TRUE, COSTS FALSE, SUMMARY true)
  SELECT * FROM postflightreports WHERE tlborder IS NOT NULL;

-- Title: Retrieve all postflightreports without technicallogbookorders
EXPLAIN (ANALYZE TRUE, COSTS FALSE, SUMMARY true)
  SELECT * FROM postflightreports WHERE tlborder IS NULL;

-- Title: Retrieve all postflightreports and their corresponding technicallogbookorders
EXPLAIN (ANALYZE TRUE, COSTS FALSE, SUMMARY true)
  SELECT *
  FROM postflightreports pfr
  JOIN technicallogbookorders tlb ON pfr.tlborder = tlb.workorderid;

--=================================================================================================
--=================================== Implementation with an FK in technicallogbookorders ============================
DROP TABLE IF EXISTS technicallogbookorders CASCADE;
DROP TABLE IF EXISTS postflightreports CASCADE;
```