PostgreSQL

Lessons learned

BearingPoint®

PostgreSQL Page Layout

Large Text and JSON

- 8KB Pages as internal data structure: little bit of metadata + payload
- Payload > 2KB → TOAST (The Oversized-Attribute Storage Technique)
  - Tries to compress data first → if not enough → break data into multiple chunks and move to TOAST table → point to toast row
  - Up to 1GB (TEXT, JSON(B), …)
- Maximal rows: 2^32 (OID overflow)
- https://hakibenita.com/sql-medium-text-performance
- https://pganalyze.com/blog/5mins-postgres-jsonb-toast

Create test data

How to create a huge amount of data

- First of all: Create realistic test data as soon as possible → Surprising moments in which what was thought to work no longer works
  - Scalability rule: D-I-D: Design for 20x capacity. Implement for 3x capacity. Deploy for ~1.5x capacity.
- Create test data with INSERT will take you decades …
- Lock table and use COPY (https://www.postgresql.org/docs/current/sql-copy.html)
- 10MBit of internet upload was my limitation … Also C# optimizations were needed … 100.000 record chunks

```
private static List<RequestInfo> InsertRequestIds(NpgsqlConnection connection, int startWithRequestId)
{
    const string query = "COPY productinventory.request_info (id, request_id, request_timestamp, sender) FROM STDIN (FORMAT BINARY)";

    int recordsWritten = 0;
    int recordsWrittenTotal = 0;

    var requestInfos = new List<RequestInfo>(RecordsPerRun * RequestsPerProduct);

    var requestInfoWriter = connection.BeginBinaryImport(query);
```

```
private static void WriteRequestInfo(NpgsqlBinaryImporter requestInfoWriter, RequestInfo requestInfo)
{
    requestInfoWriter.StartRow();
    requestInfoWriter.Write(requestInfo.RequestId, NpgsqlTypes.NpgsqlDbType.Integer);
    requestInfoWriter.Write(RandomString(30), NpgsqlTypes.NpgsqlDbType.Varchar);
    requestInfoWriter.Write(requestInfo.RequestTimeStamp, NpgsqlTypes.NpgsqlDbType.Timestamp);
    requestInfoWriter.Write("data_load_generator", NpgsqlTypes.NpgsqlDbType.Varchar);
}
```

Shared cluster

Lots of projects on the same PostgreSQL instance

- You wake up in the morning and want to continue your work. You execute the statement – which took 10ms yesterday – again → 10 seconds
- You don’t understand what's going on and re-execute it → constant 10ms
- Reason: There are 30 projects on the cluster and you were kicked out of the cache
- In SQL Server you can control resources more precise: https://learn.microsoft.com/en-us/sql/relational-databases/resource-governor/resource-governor?view=sql-server-ver16
- PostgreSQL you need multiple instances on different ports (Check out connection pooler https://www.pgbouncer.org/)

Dealing with slow queries

How to find them

- In the config file: log_min_duration_statement = 5000 (or for a single database instance: ALTER DATABASE test SET log_min_duration_statement = 5000)
  - Queries slower than 5 seconds will be written to the log file
- EXPLAIN (ANALYZE ON, BUFFERS ON) will help to understand slow queries: Was the index used? Was there a cache miss? Which join method was used? Nested-Loop-Join / Hash-Join / … ?
  - See https://www.cybertec-postgresql.com/en/join-strategies-and-performance-in-postgresql/
- auto_explain can be enabled → will also write execution plan for slow queries to the log file
- Also helpful is to watch pg_stat_statements

Index

How the deal with table scans

- PostgreSQL has several index types: B-tree, Hash, GiST, SP-GiST, GIN and BRIN
- For “columns” / “normal” operators (like =) we use B-trees: SELECT world.foo FROM bar WHERE prop = 1234
  - We create an index on column “prop”: CREATE INDEX foo_prop_idx ON world.foo(prop)
- See https://habr.com/en/company/postgrespro/blog/443284/
- Nice features used:
  - Partitial Index: CREATE INDEX foo_prop_idx ON world.foo(prop) WHERE is_deleted_by_request_id IS NOT NULL AND name IN (‘A’, ‘B’) → reduce size
    - Example: Characteristics (Key / Value) where only some need to be searchable
  - Multicolumn index: SELECT name FROM world.foo WHERE major = 1 AND minor = 2 → CREATE INDEX foo_major_minor_idx ON world.foo(major, minor);
    - Just using minor won’t work
  - Unique index: CREATE UNIQUE INDEX name ON table (column [, ...]);
  - Include columns: CREATE UNIQUE INDEX title_idx ON films (title) INCLUDE (director, rating); → SELECT director, rating → Index-Scan-Only
    - See https://www.postgresql.org/docs/current/sql-createindex.html and https://use-the-index-luke.com/blog/2019-04/include-columns-in-btree-indexes

Indizes cont.

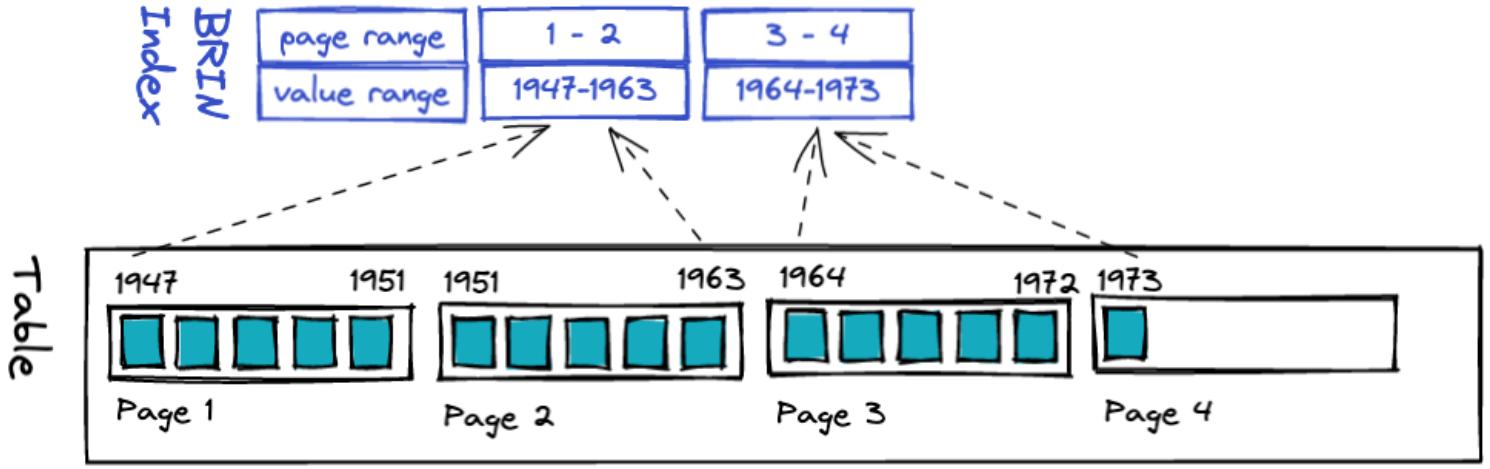
What competitors offer: SQL Server

- SQL Server
  - Clustered Index: How is the tables sorted? A table can have just on clustered index …
  - Non-Clustered Index: A table can have multiple
    - Implemented as B-Tree
  - Columstore-Index
    - Lots of repeating content which should be aggregated

BRIN Index

If you want to save looooooots of storage and gain some speed

- “BRIN stands for Block Range Index. BRIN is designed for handling very large tables in which certain columns have some natural correlation with their physical location within the table.”
- Imagine sensor data with ascending timestamp
- Btree Index: ~850MB, BRIN: ~95kB
- BRIN stores min / max of pages
- https://www.crunchydata.com/blog/avoiding-the-pitfalls-of-brin-indexes-in-postgres



BearingPoint

GiST index

Some use cases

- Why GiST? Standard B-Tree only works with < and <= and = and >= and > with standard types
- But modern database also store geodata, text documents, images, … - or you want other operators (overlap, distance, …)
- Nice for overlapping stuff
  - ALTER TABLE some_table_history ADD CONSTRAINT some_table_history_id_tstzrange_excl EXCLUDE USING gist(id WITH =, tstzrange(history_record_valid_from, history_record_valid_to) WITH &&); → records with the same (=) id should not overlap (&&)
  - More advanced:

```
=> CREATE TABLE zoo (
  cage   INTEGER,
  animal TEXT,
  EXCLUDE USING GIST (cage WITH =, animal WITH <>)
);

=> INSERT INTO zoo VALUES(123, 'zebra');
INSERT 0 1
=> INSERT INTO zoo VALUES(123, 'zebra');
INSERT 0 1
=> INSERT INTO zoo VALUES(123, 'lion');
ERROR:  conflicting key value violates exclusion constraint "zoo_cage_animal_excl"
DETAIL:  Key (cage, animal)=(123, lion) conflicts with existing key (cage, animal)=(123,
=> INSERT INTO zoo VALUES(124, 'lion');
INSERT 0 1
```

```
CREATE TABLE car_reservation (
car text,
during tsrange,
EXCLUDE USING GIST (car WITH =, add_buffer(during, '1 hours'::interval) WITH &&)
);
```

```
CREATE TABLE test (a int4);
-- create index
CREATE INDEX testidx ON test USING GIST (a);
-- query
SELECT * FROM test WHERE a < 10;
-- nearest-neighbor search: find the ten entries closest to "42"
SELECT *, a <-> 42 AS dist FROM test ORDER BY a <-> 42 LIMIT 10;
```

Row identifiers

Then and now

- id SERIAL PRIMARY KEY → creates a sequence
  - make schema, dependency, and permission management unnecessarily complicated
- NEW since PostgreSQL 10: id INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY → “hided” behind the table
- See https://www.enterprisedb.com/blog/postgresql-10-identity-columns-explained
- Support by Hibernate:

```
@Data
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "persons")
public class Person {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
```

PostgreSQL and time

Use the correct type

- Never ever use “TIMESTAMP WITHOUT TIME ZONE” → name is misleading: Don’t make the mistake: “I just want to store UTC”
- Always (!) use “TIMESTAMP WITH TIME ZONE”
  - PostgreSQL won’t store the time zone → you have to store it in a separate column
  - BUT: PostgreSQL will respect the time zone at the time of INSERT / UPDATE
  - Example:
    - SHOW TIMEZONE; → Europe/Berlin (+01:00)
    - SELECT '2009-01-01T12:00:00+05:00'::TIMESTAMP → 2009-01-01T11:00:00.0Z
    - SELECT '2009-01-01T12:00:00+05:00'::TIMESTAMPTZ → 2009-01-01T07:00:00.0Z
  - Summary:
    - TIMESTAMP (WITHOUT TIMEZONE): Will just ignore the timezone → just takes the raw date/time → just there for legacy → no reason to use
    - TIMESTAMPTZ means: respect timezone but it **will be always stored as UTC PostgreSQL internally** (and the timezone information is lost)

Don’t use MONEY type

How to store amounts of money

- Internally fixed-point type, implemented as a machine int, doesn't store a currency
- NUMERIC(15,6) → 15 in total, 6 right of the decimal point
- Store currency speratly

Hibernate and magic queries

Some solution to deal with queries

- Its actually not a fault of Hibernate itself → sometimes wrong expectations, wrong annotations
  - On the other side: Stackoverflow itself use MicroORMs: https://github.com/DapperLib/Dapper → All queries are written by hand + verified by hand
- But how ensure consistency? Created a rule: Critical path of the queries are written in SQL, it must be reviewed on a system with realistic data
- Pitfalls: Business Logic comes to the database
  - Example: Loading airplane (how many cargo containers can be put onto the plan?). Concurrency may be handled in persistence → but then some business logic is in persistence
- Example:

```
@Query(nativeQuery = true, value = "SELECT schema.get_related_products_by_customer_id(:customerId)")
LinkedHashSet<Integer> getRelatedProductsByCustomerId(UUID customerId);
```

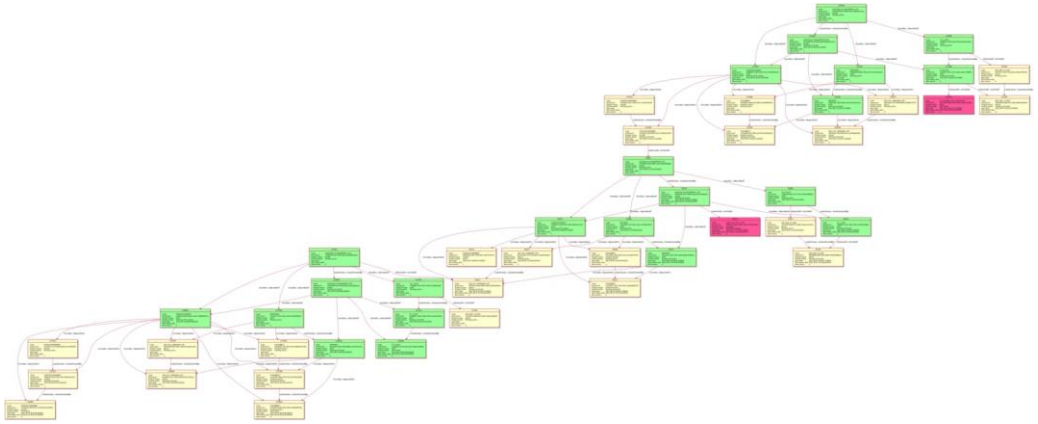
Data historization

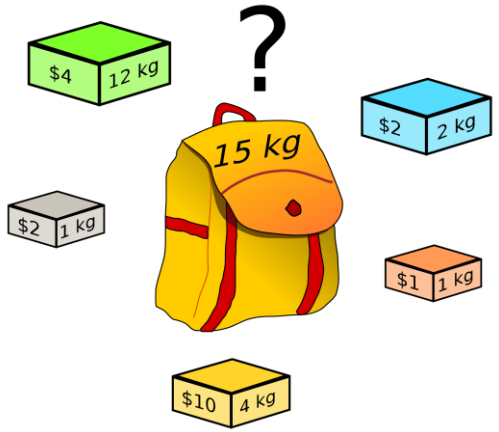
Why build it on your own

- There are great tools out there: https://hibernate.org/orm/envers/
- But requirements may look like:
  - A request changes the data. So before the request we have “data old” – after the request “data new”
    - There should be a persistence constraint on that
    - History records are not allowed to overlap → forced by persistence
  - A request has a timestamp → the timestamp should be used in all history tables to get a consistent view of the data
    - So we have “some_table” and “some_table_history” which has columns “history_record_valid_from” and “history_record_valid_to”
- Why separated tables?
  - History tables can be optimized in a different way and have duplicate “id” columns (since a record can change multiple times)
  - History tables are readonly for normal users / application users
- Triggers are used to write data to the history table
  - Something like https://postgis.net/workshops/postgis-intro/history_tracking.html
- Other DBs call it “temporal tables” → see https://www.sqlshack.com/temporal-tables-in-sql-server/
- Sample implementation: https://github.com/mvodep/Playground/tree/main/Hibernate%20History

Pgday: Solving the knapsack problem with recursive queries and PostgreSQL

- Recursive CTEs were used to solve the problem → See https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-recursive-query/
- Talk can be seen here: https://www.youtube.com/watch?v=njvH3I39Dv0
- Some syntactic sugar was added in PostgreSQL 14: https://aiven.io/blog/explore-the-new-search-and-cycle-features-in-postgresql-14
- If the graph is small – it works very well → if it gets a little bit bigger and complex (e.g. depth = 13) – will run quite long (we talk about minutes) → because JOIN tables grow and grow
- From a project: Iterate approaches may also work → I was not able to run the graph below with recursive CTE → several hours of research





BearingPoint

PostgreSQL and Graphs

Analyse the need and don’t see only the word “graph”

- If you hear the term “person graph” you may think: “I need a graph database”
- RDBMS works int terms of rows → but now we need a graph
- Project experience:
  - You may have a m:n table: persons and person_2_person → Lots of clusters with about 100 persons
  - But why do we calculate the hierarchy all the time?
  - Other solution: “person_graph” and all persons link to it → so the structure of the graph doesn't matter any more
- Strength of RDBMS: Find rows → can be easily formulated with a graph id
- Take away: understand the problem, then decide for a technology

Pgday: ORM IS BAD – CAN WE OFFER AN ALTERNATIVE?

Return JSON from the database

- Talk: https://www.youtube.com/watch?v=ed6d_aSsIcA
- There were some scripts which scan tables and generate functions for generating responses as JSON
- So the ORM moves more or less to the database
- My thoughts:
  - For transferring big datasets – maybe a problem?
  - Also the questions at the end showed, that people were confused by that approach → but in the end of the day: its worth to know the approach

Pgday: Wie schaffe ich 1000 Applikationsuser gleichzeitig?

20 connections are enough

- Take away: Connections are not cheap → so they should kept open
- Pooling can be done in
  - JAVA https://github.com/brettwooldridge/HikariCP
  - External service: https://www.pgbouncer.org/ https://www.pgpool.net/mediawiki/index.php/Main_Page
    - Some types of LDAP can be problematic with this approach
- Customers were confused when the consultant answers “20 Connections are enough”
- Hugest challenge: Planning / Finding the correct numbers → some hints https://www.cybertec-postgresql.com/en/estimating-connection-pool-size-with-postgresql-database-statistics/
- https://wiki.postgresql.org/wiki/Number_Of_Database_Connections

Pgday: pg_hint_plan – get the right plan without surprises

For hard core users

- Hints for the planner can be written as comment above the query
- Assumption: Users knows the data better than the statistics can

Pgday: POSTGRES PITFALLS

- Take away: always use schema in front of functions (foobar.my_func(…)) or tables (SELECT a FROM foobar.my_table) → a little bit of speed gain + security problems when function exists in other schemas and default search path is not as
- Some points from here: https://www.cybertec-postgresql.com/en/postgresql-security-things-to-avoid-in-real-life/

Pgday: Praktische Transaktionstheorie für PostgreSQL-Anwender

Concurrency

- Project experience: Optimistic concurrency (rowversion) works will with well defined aggregates
- If it goes deeper: very, very complicated topic … Consistency over several rows is a complicated topic → deep understanding needed
  - https://www.cybertec-postgresql.com/en/postgresql-constraints-over-multiple-rows/
  - https://www.cybertec-postgresql.com/en/lock-table-can-harm-your-database/
  - https://www.cybertec-postgresql.com/en/triggers-to-enforce-constraints/
  - https://www.cybertec-postgresql.com/en/transaction-anomalies-with-select-for-update/
  - And so on

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read	Serialization Anomaly
Read uncommitted	Allowed, but not in PG	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible	Possible
Repeatable read	Not possible	Not possible	Allowed, but not in PG	Possible
Serializable	Not possible	Not possible	Not possible	Not possible

General: Schema versioning with liquibase

Just an observation from a project

- Worked well – we used the native SQL strategy
- At the beginning we wrote rollbacks script – after the first data migration we gave up (rollback = would be hard to get the data back how it was before)
- Some thoughts:
  - Some huge migration caused k8s to kill the pod over and over → had to do the migration manually
  - At the end of the day: Script maintenance would be also nice → meanwhile a long story of changes – snapshot?

Debezium

- Outbox pattern to ensure, that events are published
  - Several implementations: for example debezium as worker (so raw content is written to outbox table)
- Problems occur when PostgreSQL cluster fail over occurs: replication slots die …
  - See docs: There must be a process that re-creates the Debezium replication slot before allowing the application to write to the new primary. This is crucial. Without this process, your application can miss change events.
  - Solved in 2022 by Patroni: https://www.percona.com/blog/how-patroni-addresses-the-problem-of-the-logical-replication-slot-failover-in-a-postgresql-cluster/

General: Personal Talks at PgDay

- Never use ZFS as PostgreSQL filesystem → Snapshots slow it down (Cybertec employee)
- Customer often request Master-Master replication → in 90% of the cases requirements are analysed wrong → Sharding
- Observations: Lots (5) of companies present - they offer cloud-hosted-solutions
- https://www.cybertec-postgresql.com/de/produkte/cybertec-migrator/ is now free available – talked to some dev → helps migrate from Oracle → PostgreSQL
- Talked to a cybertec developer: They maintain petabytes-PostgreSQL-databases
- “If we had this data what would it mean?” https://www.cybertec-postgresql.com/en/sql-trickery-hypothetical-aggregates/

PACELC

- network partitioning (P) in a distributed computer system, one has to choose between availability (A) and consistency (C) (as per the CAP theorem)
- but else (E), even when the system is running normally in the absence of partitions, one has to choose between latency (L) and consistency (C).

DDBS	P+A	P+C	E+L	E+C
BigTable/HBase		✓		✓
Cassandra	✓		✓[a]	
Cosmos DB		✓	✓ [b]	
Couchbase		✓	✓	✓
DynamoDB	✓		✓[a]	
FaunaDB[8]		✓	✓	✓
Hazelcast IMDG[5][6]	✓	✓	✓	✓
Megastore		✓		✓
MongoDB	✓			✓
MySQL Cluster		✓		✓
PNUTS		✓	✓	
PostgreSQL		✓		✓
Riak	✓		✓[a]	
VoltDB/H-Store		✓		✓

RDBMS are dead?

NoSQL is the future?

- Definitely not!
  - They can live side by side → NoSQL is a enrichment of your available tools …
- If a document is deeply nested → You will get lots of tables …
  - On the one hand: In RDBMS you can specify lots of constraints, fine tune everything → but “some” complexity overhead
  - On the other hand: Define a schema if you want and just store it
- As always: It depends on the use case / requirements ;-)
  - But would be also a nice talk …

BearingPoint®