



Data warehousing with PostgreSQL

Audience

- Case of one PostgreSQL node data warehouse
 - This talk does not directly address multi-node distribution of data
- Limitations on disk usage and concurrent access
 - No rule of thumb
 - Depends on a careful analysis of data flows and requirements
- Small/medium size businesses



Summary

- Data warehousing introductory concepts
- PostgreSQL strengths for data warehousing
- Data loading on PostgreSQL
- Analysis and reporting of a PostgreSQL DW
- Extending PostgreSQL for data warehousing
- PostgreSQL current weaknesses

Part one: Data warehousing basics

- Business intelligence
- Data warehouse
- Dimensional model
- Star schema
- General concepts



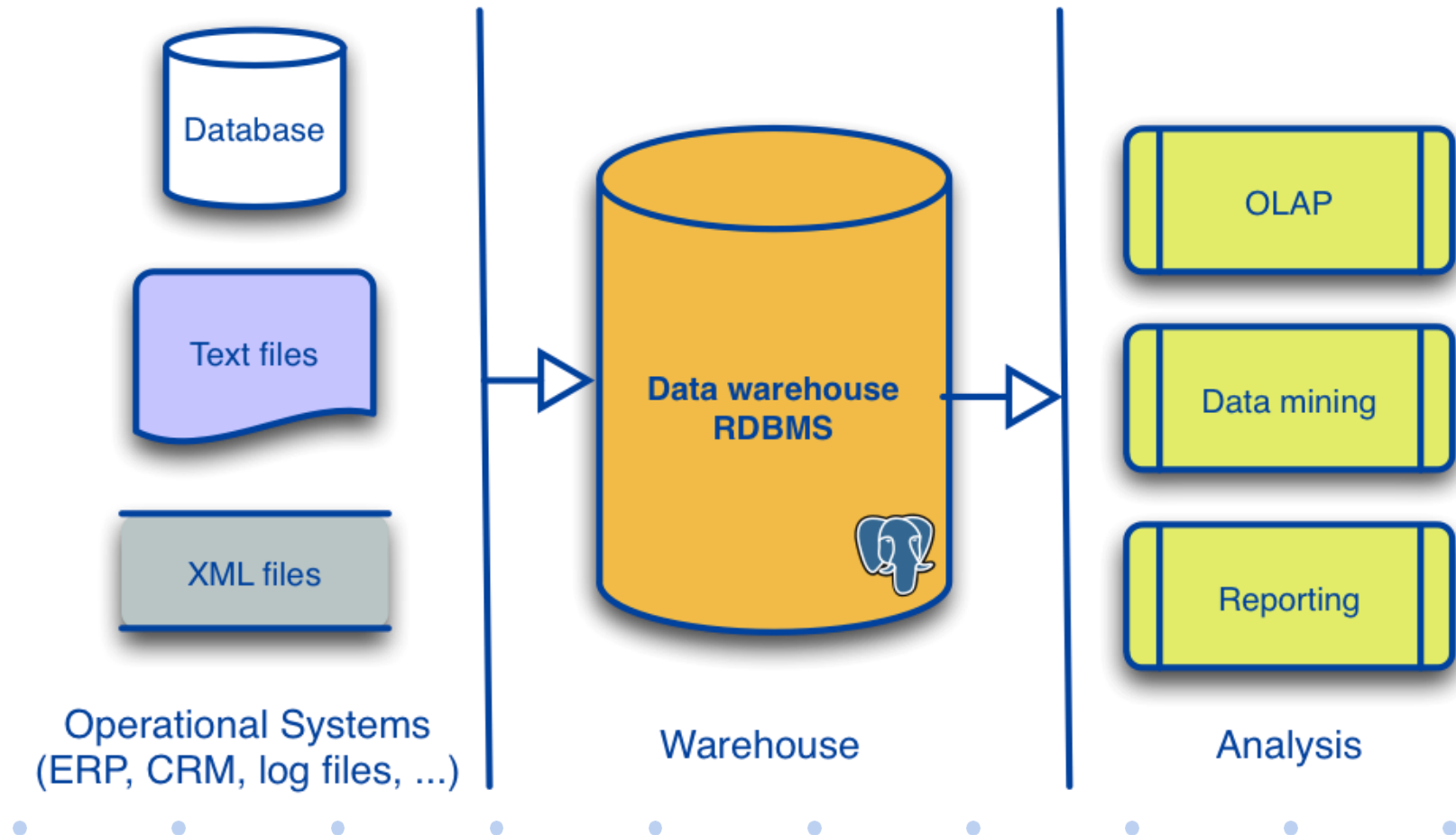
Business intelligence & Data warehouse



- **Business intelligence:** *“skills, technologies, applications and practices used to help a business acquire a better understanding of its commercial context”*
- **Data warehouse:** *“A data warehouse houses a standardized, consistent, clean and integrated form of data sourced from various operational systems in use in the organization, structured in a way to specifically address the reporting and analytic requirements”*
 - *Data warehousing is a broader concept*



A simple scenario



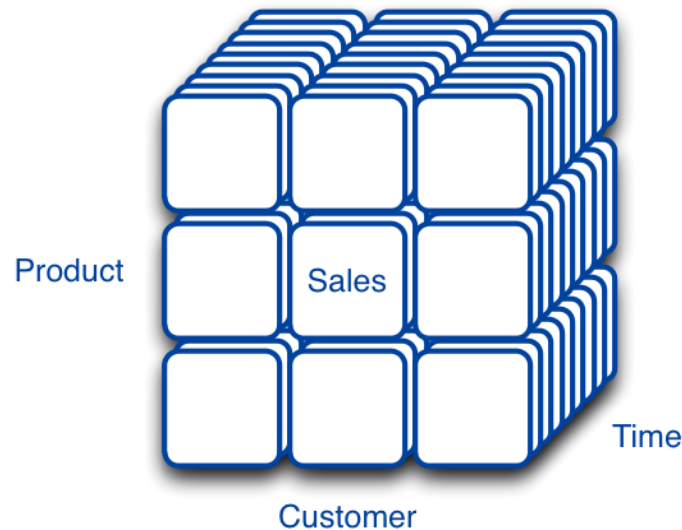
PostgreSQL = RDBMS for DW?

- The typical storage system for a data warehouse is a Relational DBMS
- Key aspects:
 - Standards compliance (e.g. SQL)
 - Integration with external tools for loading and analysis
- PostgreSQL 8.4 is an ideal candidate

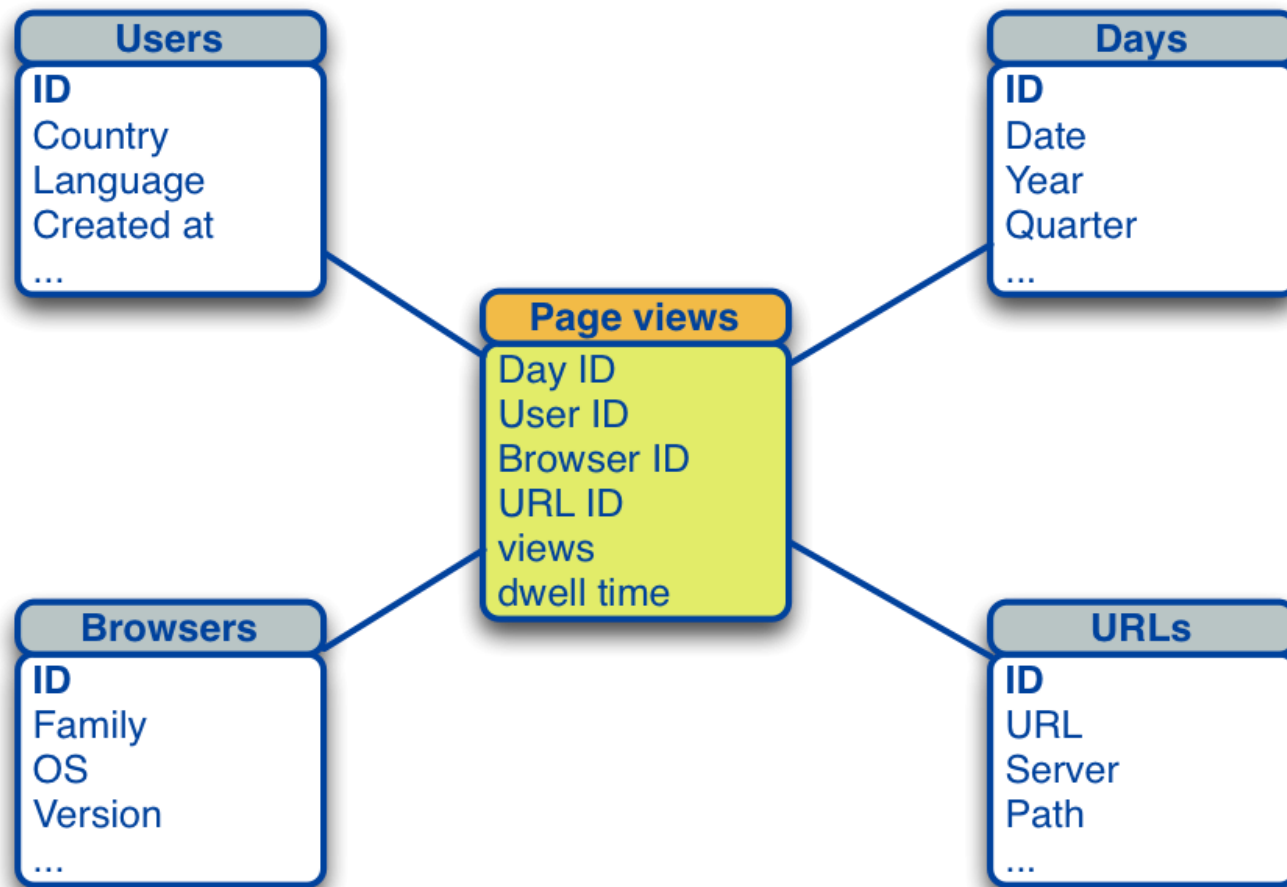


Example of dimensional model

- Subject: commerce
- Process: sales
- Dimensions: customer, product
 - Analyse sales by customer and product over time



Star schema



General concepts

- Keep the model simple (star schema is fine)
- Denormalise tables
- Keep track of changes that occur over time on dimension attributes
- Use **calendar tables** (static, read-only)



Example of calendar table

```
-- Days (calendar date)
CREATE TABLE calendar (
    -- days since January 1, 4712 BC
    id_day INTEGER NOT NULL PRIMARY KEY,
    sql_date DATE NOT NULL UNIQUE,
    month_day INTEGER NOT NULL,
    month INTEGER NOT NULL,
    year INTEGER NOT NULL,
    week_day_str CHAR(3) NOT NULL,
    month_str CHAR(3) NOT NULL,
    year_day INTEGER NOT NULL,
    year_week INTEGER NOT NULL,
    week_day INTEGER NOT NULL,
    year_quarter INTEGER NOT NULL,
    work_day INTEGER NOT NULL DEFAULT '1'
    ...
);
```

Part two: PostgreSQL and DW

- General features
- Stored procedures
- Tablespaces
- Table partitioning
- Schemas / namespaces
- Views
- Windowing functions and WITH queries



General features

- Connectivity:
 - PostgreSQL perfectly integrates with external tools or applications for data mining, OLAP and reporting
- Extensibility:
 - User defined data types and domains
 - **User defined functions**
 - Stored procedures



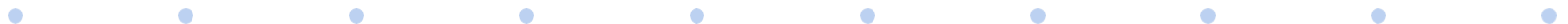
Stored Procedures

- Key aspects in terms of data warehousing
- Make the data warehouse:
 - flexible
 - intelligent
- Allow to analyse, transform, model and deliver data within the database server



Tablespaces

- Internal label for a physical directory in the file system
- Can be created or removed at anytime
- Allow to store objects such as tables and indexes on different locations
- Good for scalability
- Good for performances

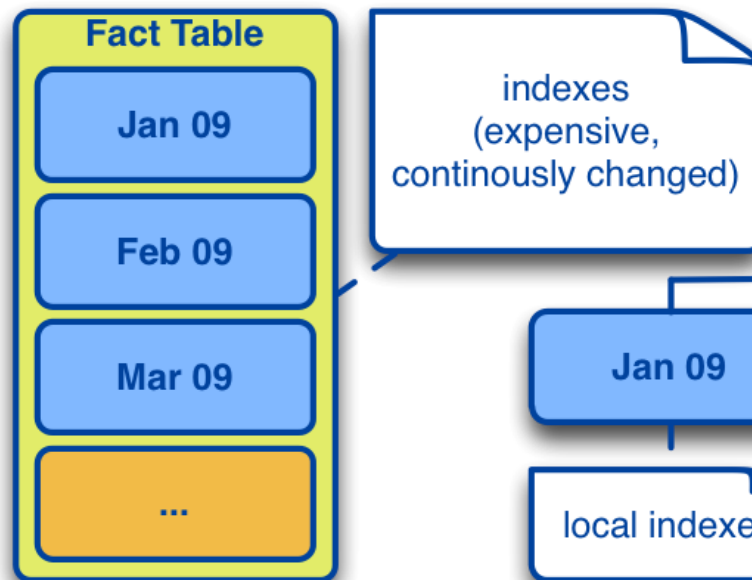


Horizontal table partitioning

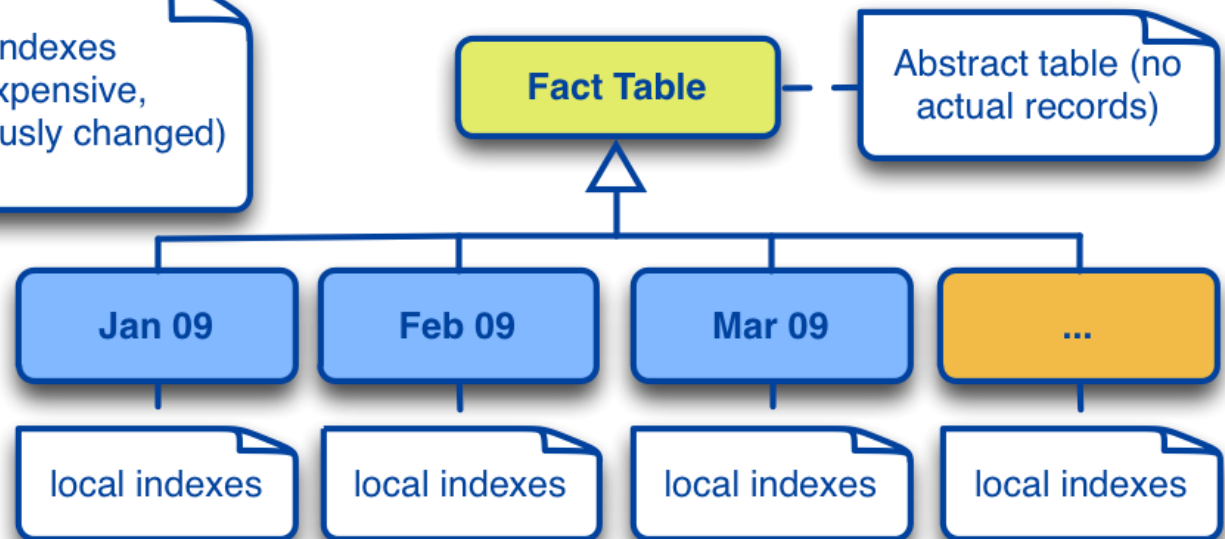
1/2

- A physical design concept
- Basic support in PostgreSQL through inheritance

single large table scenario



horizontal partitioning scenario with inheritance



Views and schemas

- Views:
 - Can be seen as “*placeholders*” for queries
 - PostgreSQL supports read-only views
 - Handy for summary navigation of fact tables
- Schemas:
 - Similar to the “*namespace*” concept in OOA
 - Allows to organise database objects in logical groups

Window functions and WITH queries

- Both added in PostgreSQL 8.4
- Window functions:
 - perform aggregate/rank calculations over partitions of the result set
 - more powerful than traditional “GROUP BY”
- WITH queries:
 - label a *subquery block*, execute it once
 - allow to reference it in a query
 - can be recursive



Part three: Optimisation techniques

- Surrogate keys
- Limited constraints
- Summary navigation
- Horizontal table partitioning
- Vertical table partitioning
- “Bridge tables” / Hierarchies



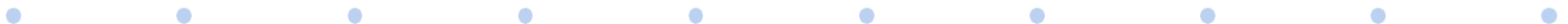
Use surrogate keys

- Record identifier within the database
- Usually a sequence:
 - `serial` (INT sequence, 4 bytes)
 - `bigserial` (BIGINT sequence, 8 bytes)
- Compact primary and foreign keys
- Allow to keep track of changes on dimensions



Limit the usage of constraints

- Data is already consistent
- No need for:
 - referential integrity (foreign keys)
 - check constraints
 - not-null constraints



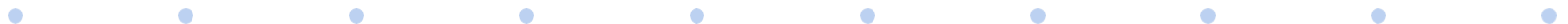
Implement summary navigation

- Analysing data through hierarchies in dimensions is very time-consuming
- Sometimes *caching* these summaries is necessary:
 - real-time applications (e.g. web analytics)
 - can be achieved by simulating materialised views
 - requires careful management on latest appended data
 - Skytools' PgQ can be used to manage it
- Can be totally delegated to OLAP tools



Horizontal (table) partitioning

- Partition tables based on record characteristics (e.g. date range, customer ID, etc.)
- Allows to split fact tables (or dimensions) in smaller chunks
- Great results when combined with tablespaces



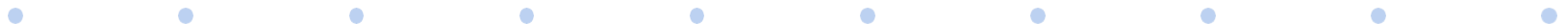
Vertical (table) partitioning

- Partition tables based on columns
- Split a table with many columns in more tables
- Useful when there are fields that are accessed more frequently than others
- Generates:
 - Redundancy
 - Management headaches (careful planning)



Bridge hierarchy tables

- Defined by Kimball and Ross
- Variable depth hierarchies (flattened trees)
- Avoid recursive queries in parent/child relationships
- Generates:
 - Redundancy
 - Management headaches (careful planning)



Example of bridge hierarchy table

```
id_bridge_category | integer | not null
category_key       | integer | not null
category_parent_key | integer | not null
distance_level     | integer | not null
bottom_flag        | integer | not null default 0
top_flag           | integer | not null default 0
```

id_bridge_category	category_key	category_parent_key	distance_level	bottom_flag	top_flag
1	1	1	0	0	1
2	586	1	1	1	0
3	587	1	1	1	0
4	588	1	1	1	0
5	589	1	1	1	0
6	590	1	1	1	0
7	591	1	1	1	0
8	2	2	0	0	1
9	3	2	1	1	0

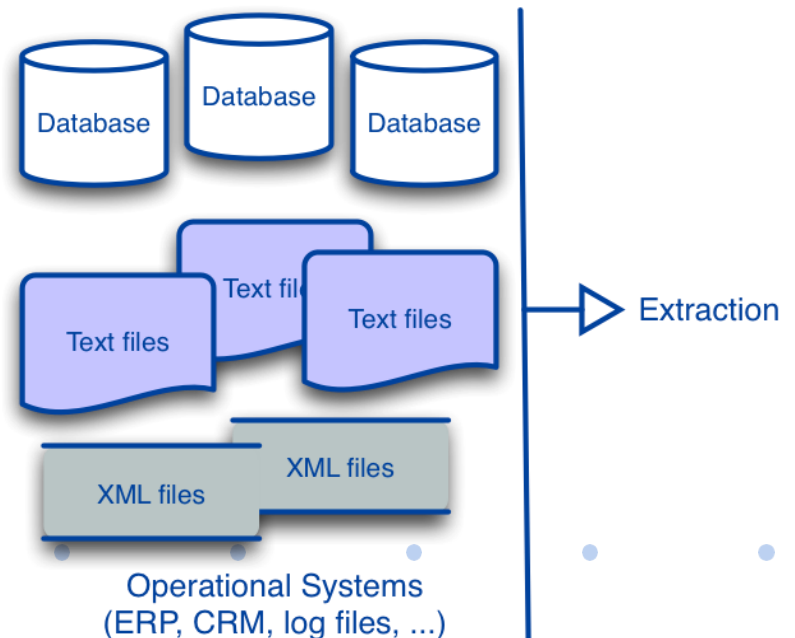
Part four: Data loading

- Extraction
- Transformation
- Loading
- ETL or ELT?
- Connecting to external sources
- External loaders
- Exploration data marts



Extraction

- Data may be originally stored:
 - in different locations
 - on different systems
 - in different formats (e.g. database tables, flat files)
- Data is extracted from source systems
- Data may be filtered



Transformation

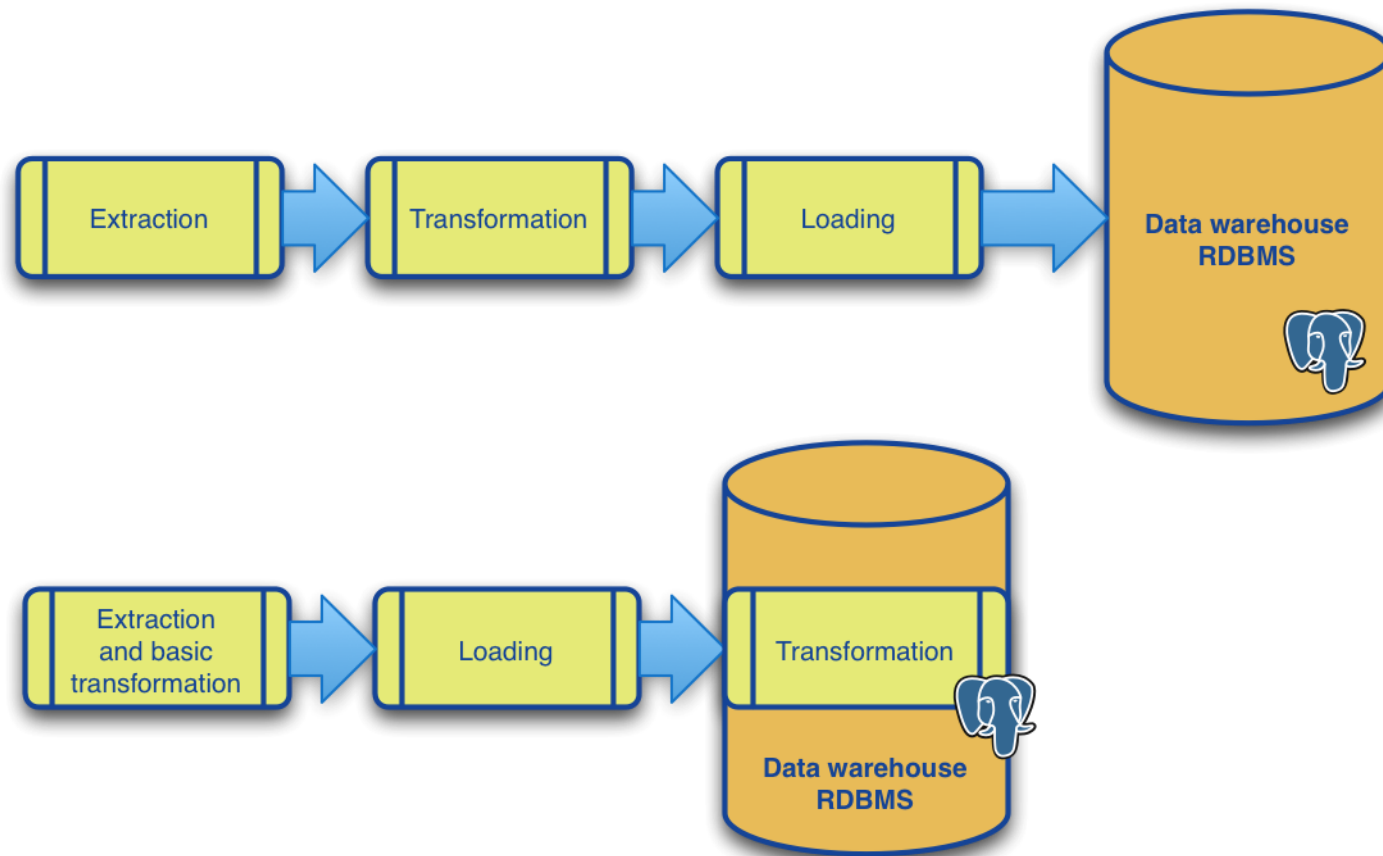
- Data previously extracted is transformed
 - Selected, filtered, sorted
 - Translated
 - Integrated
 - Analysed
 - ...
- Goal: prepare the data for the warehouse

Loading

- Data is loaded in the warehouse database
- Which frequency?
- Facts are usually appended
 - Issue: aggregate facts need to be updated



ETL or ELT?



Connecting to external sources

- PostgreSQL allows to connect to external sources, through some of its extensions:
 - dblink
 - PL/Proxy
 - DBI-Link (any database type supported by Perl's DBI)
- External sources can be seen as database tables
- Practical for ETL/ELT operations:
 - `INSERT ... SELECT` operations



External tools

- External tools for ETL/ELT can be used with PostgreSQL
- Many applications exist
 - Commercial
 - Open-source
 - Kettle (part of Pentaho Data Integration)
- Generally use ODBC or JDBC (with Java)

Exploration data marts

- Business requirements change, continuously
- The data warehouse must offer ways:
 - to explore the historical data
 - to create/destroy/modify data marts in a staging area
 - connected to the production warehouse
 - totally independent, safe
 - this environment is commonly known as *Sandbox*

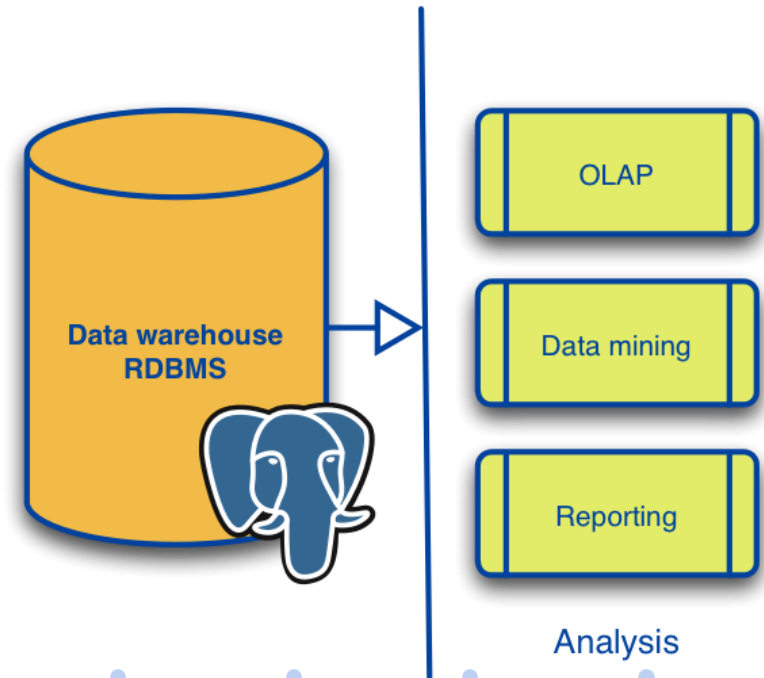


Part five: Beyond PostgreSQL

- Data analysis and reporting
- Scaling a PostgreSQL warehouse with PL/Proxy

Data Analysis and reporting

- Ad-hoc applications
- External BI applications
 - Integrate your PostgreSQL warehouse with third-party applications for:
 - OLAP
 - Data mining
 - Reporting
 - Open-source examples:
 - Pentaho Data Integration



Scaling with PL/Proxy

- PL/Proxy can be directly used for querying data from a single remote database
- PL/Proxy can be used to speed up queries from a local database in case of multi-core server and partitioned table
- PL/Proxy can also be used:
 - to distribute work on several servers, each with their own part of data (known as *shards*)
 - to develop *map/reduce* type analysis over sets of servers



Part six: PostgreSQL's weaknesses

- Native support for data distribution and parallel processing
- On-disk bitmap indexes
- Transparent support for data partitioning
- Transparent support for materialised views
- Better support for “temporal” needs



Data distribution & parallel processing

- Shared nothing architecture
- Allow for (massive) parallel processing
- Data is partitioned over servers, in shards
- PostgreSQL also lacks a `DISTRIBUTED BY` clause
- PL/Proxy could potentially solve this issue



On-disk bitmap indexes

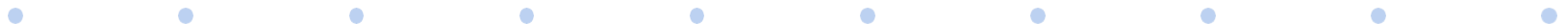
- Ideal for data warehouses
- Use bitmaps (vectors of bits)
- Would perfectly integrate with PostgreSQL in-memory bitmaps for **bitwise logical operations**

ID	Gender	Bitmaps	
		F	M
1	Female	1	0
2	Male	0	1
3	Male	0	1
4	NULL	0	0
5	Female	1	0

Source: Wikipedia

Transparent table partitioning

- Native transparent support for table partitioning is needed
 - `PARTITION BY` clause is needed
 - Partition daily management



Materialised views

- Currently can be simulated through stored procedures and views
- A transparent native mechanism for the creation and management of materialised views would be helpful
 - Automatic Summary Tables generation and management would be cool too!



Temporal extensions

- Some of TSQL2 features could be useful:
 - Period data type
 - Comparison functions on two periods, such as
 - Precedes
 - Overlaps
 - Contains
 - Meets



Conclusions

- PostgreSQL is a suitable RDBMS technology for a **single node data warehouse**:
 - FLEXIBILITY
 - Performances
 - Reliability
 - Limitations apply
- For open-source **multi-node data warehouse**, use SkyTools (pgQ, Londiste and PL/Proxy)
- If **Massive Parallel Processing** is required:
 - Custom solutions can be developed using PL/Proxy
 - Easy to move up to commercial products based on PostgreSQL like Greenplum, if data volumes and business requirements need it

Recap

- Data warehousing introductory concepts
- PostgreSQL strengths for data warehousing
- Data loading on PostgreSQL
- Analysis and reporting of a PostgreSQL DW
- Extending PostgreSQL for data warehousing
- PostgreSQL current weaknesses





**The Reporting solution
complements Qlik**