HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Build your own Database

Week 7

Agenda

- Q&A Sprint 3
- Group Projects
- Logistics
- Scheduling
- MVCC

Sprint 3

Questions?

NUMA-Optimized Join

- As of now, Hyrise supports three join types (Hash, Sort-Merge, and Nested Loop)
- Most of them are optimized for parallel execution (cf. radix partitioning from last week)
- However, large modern NUMA systems move the major bottleneck from CPU caches to the NUMA bus
  - Over the past months, Hyrise gained NUMA support … let’s leverage it!

NUMA-Optimized Join

Tasks:

- Warm-Up: IndexJoin
- Implement the *massively parallel sort-merge-join* [1]
- Thoroughly benchmark, optimize, and compare against other joins
- Introduce join to optimizer

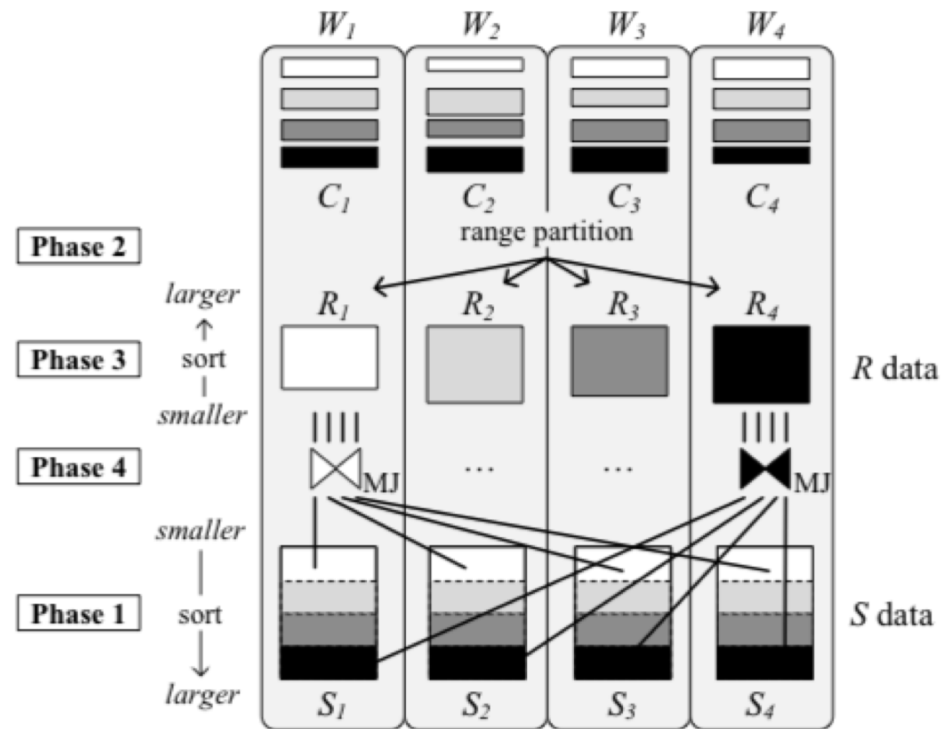


Figure 5: P-MPSM join with four workers W_i

[1] Albutiu, Massively Parallel Sort-Merge Joins in Main Memory Multi-Core Database Systems, VLDB 2012

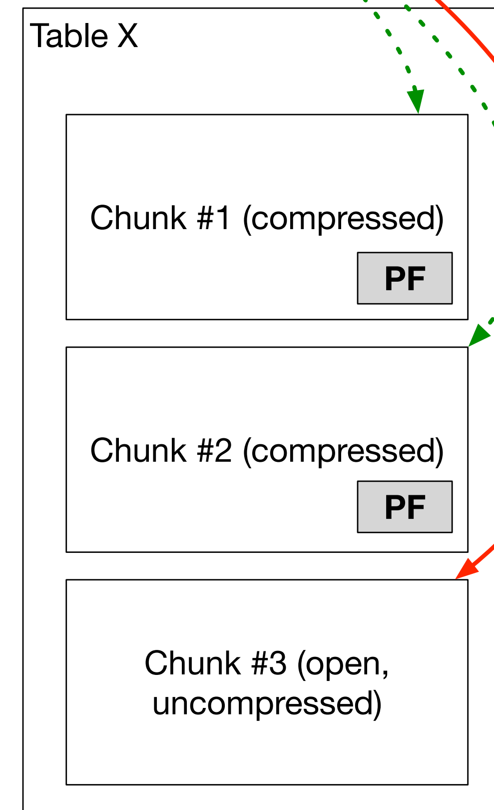
Pruning Filters

- Column scans are efficient and fast in main memory column stores, but still every avoided scan is beneficial
- This gets increasingly important for data that is not local (e.g., NUMA-distant data, data on secondary storage)
- **Goal:** we'd like to recognize unnecessary chunk accesses through small data structures (<1 MB in size; chunk-local) to improve the overall system performance

Pruning Filters

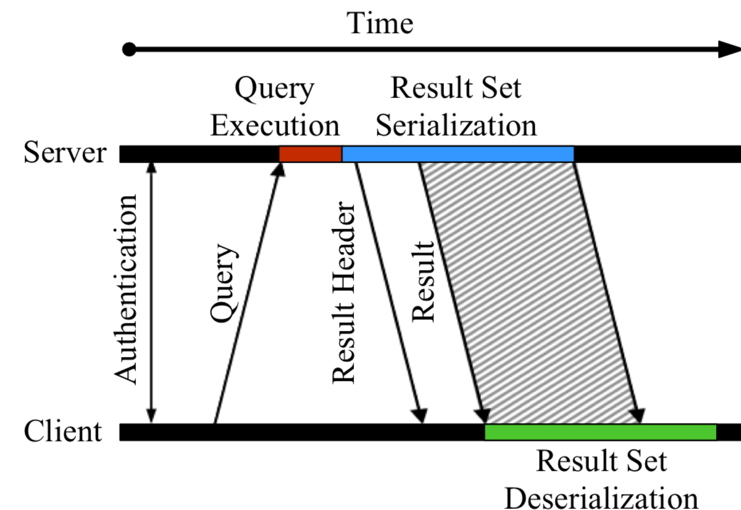
Tasks:

- Implement pruning filters in Hyrise2 with a focus on efficient creation and validation
- Introduce chunk pruning to the optimizer (+ improving estimates)
- Allow operators (e.g., joins) to prune chunks dynamically during execution



Networking

- Hyrise2 can be used as library with an interactive SQL console
- (Typically) databases
  - Are server applications, running forever,
  - Offer a **message-based protocol** for clients
  - Use TCP/IP as network protocol



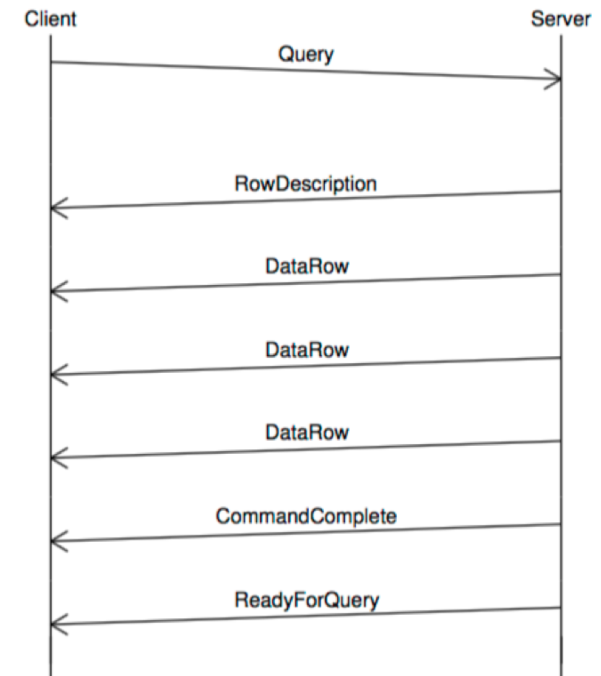
http://www.vldb.org/pvldb/vol10/p1022-muehleisen.pdf

HPI Hasso Plattner Institut

Networking

PostgreSQL message protocol:

- Regular packet: char tag, int32 len, payload



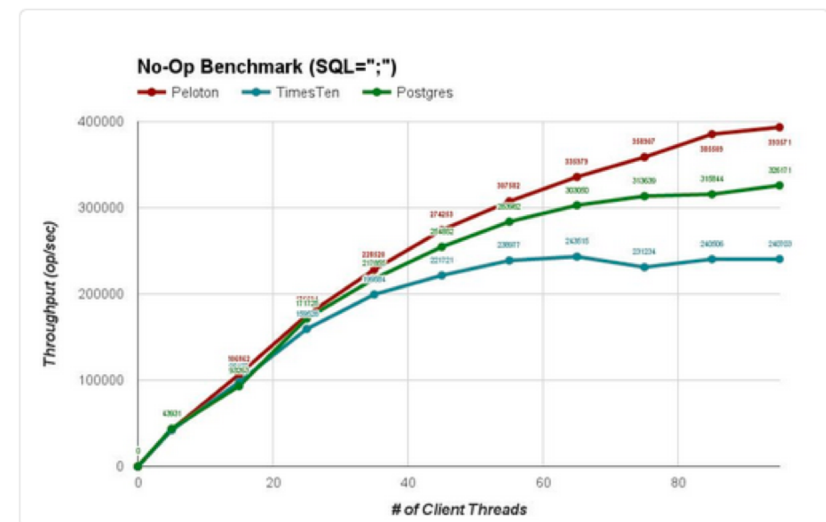
- PostgreSQL 10.1 Documentation, Chapter 52. Frontend/Backend Protocol https://www.postgresql.org/files/documentation/pdf/10/postgresql-10-A4.pdf
- https://wulczer.org/postgres-on-the-wire.pdf

Networking

Tasks:

- Define a message-based communication protocol for Hyrise2
- Integrate the protocol into the Hyrise2 console
- Implement a server
  - High throughput
  - Low overhead

Andy Pavlo @andy_pavlo · 15 Nov 2016
Our new DBMS is the fastest NOOP system available today. We can do "no work" faster than Postgres and TimesTen. This is a major breakthrough



5 20 41

Partitioning

- Is the division of relations into disjoint subsets
  - Is required if the data size exceeds the memory capacity of a single machine
  - Enables pruning (skip processing irrelevant partitions)
- Vertical vs. horizontal
- Physical vs. logical

Partitioning

- Logical view on partitioning in Hyrise2

Table #1

Partition #1 (horizontal, explicit through logical partitioning)

Chunk #1 (implicit horizontal partitioning; physical partition; possibly frozen)

Chunk #2 (implicit horizontal partitioning; physical partition; possibly open)

Partition #2 (horizontal, explicit through logical partitioning)

Chunk #3 (implicit horizontal partitioning; physical partition; possibly frozen)

Chunk #4 (implicit horizontal partitioning; physical partition; possibly open)

Partitioning

Task:

- Implement logical horizontal partitioning for Hyrise2
- If needed: Adapt operators to work with partitioned tables
- Adapt insert
- Adapt table load
- (Integrate it into SQL parser)
- (Integrate pruning into SQL optimizer)

Strings, Date(time)s and Functions

- (Enterprise) data consists of many (short) string values
- `std::string‘s` SSO is a blessing and curse

```
Mem[||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||1.01T/1.48T]
Swp[||||||||||||||||||||||||||||||                              858M/1.68G]
```

- Motivation: Footprint reduction & computational efficiency
- ~~`std::array<char, 1> ... std::array<char, 255>`~~
- Hide string size by string wrapper class

Strings, Date(time)s and Functions

- Most databases offer multiple ways to represent dates
  - `DATE`, `DATETIME` and `TIMESTAMP`
- Internal representation of date types
- (Date) functions are heavily used in benchmarks
  - `l_shipdate < date '1994-01-01' + interval '1' year`
  - `substring, dateAdd, dayOfMonth…`

HPI Hasso Plattner Institut

Strings, Date(time)s and Functions

- Step 1: Implement additional data types
- Step 2: Implement functions that build upon these data types
- Step 3: Evaluate the implementation‘s impact on benchmark performance

Self-Driving Database

- Large database deployments are configured and tuned by database administrators
  - Which indexes? How to partition data? How many threads?
- Manual tuning is difficult:
  - Large problem space
  - Inter-dependency between options
  - Decisions are tailored to specific workloads
- NP-complete optimization problems

Self-Driving Database

- **Idea:** The database knows best how to tune itself
  - It has knowledge about the underlying data, queries and their access frequencies
- This data can be fed into heuritics to generate close to optimal solutions
- Index Selection for Hyrise
  - Step 1: Extract relevant information from the query plan cache and statistics component
  - Step 2: Pass this information to selection component and create indexes accordingly

More Optimizer Rules

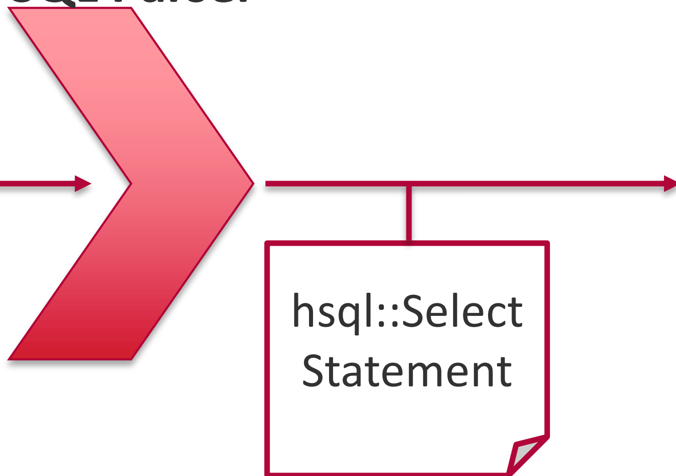
SQL Query
SELECT customer.id, COUNT(*) FROM customer LEFT JOIN order ON customer.id = order.cid WHERE order.priority = 1 GROUP BY customer.id

?

Physical Query Plan (PQP)
GetTable, TableScan, Join, Aggregate, …

More Optimizer Rules

SQL Query
SELECT customer.id, COUNT(*) FROM customer LEFT JOIN order ON customer.id = order.cid WHERE order.priority = 1 GROUP BY customer.id



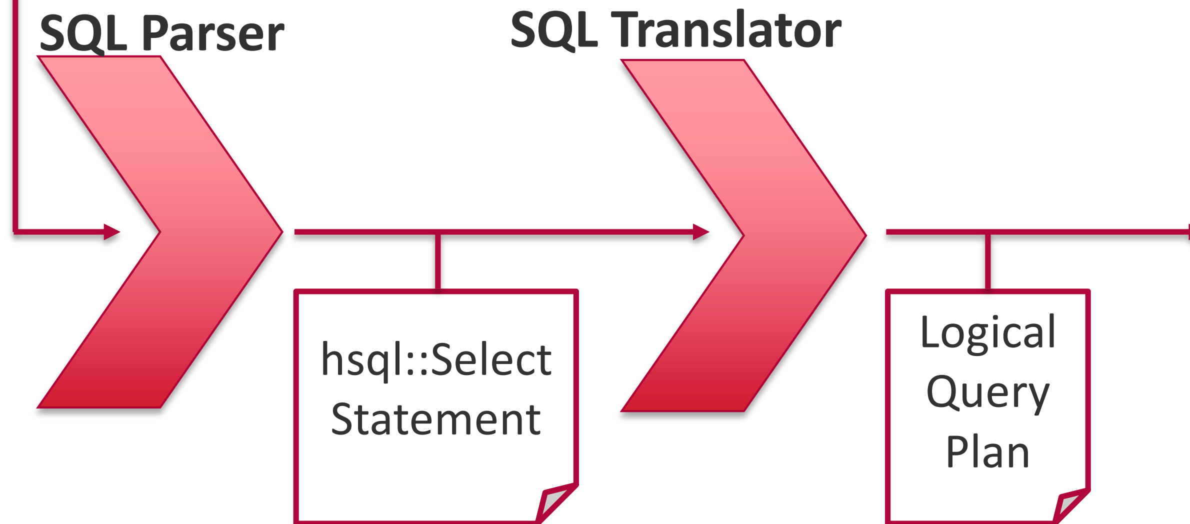

```
struct SelectStatement : SQLStatement {
   SelectStatement();
   virtual ~SelectStatement();

   TableRef* fromTable;
   bool selectDistinct;
   std::vector<Expr*>* selectList;
   Expr* whereClause;
   GroupByDescription* groupBy;

   SelectStatement* unionSelect;
   std::vector<OrderDescription*>* order;
   LimitDescription* limit;
 };
```

HPI Hasso Plattner Institut

More Optimizer Rules

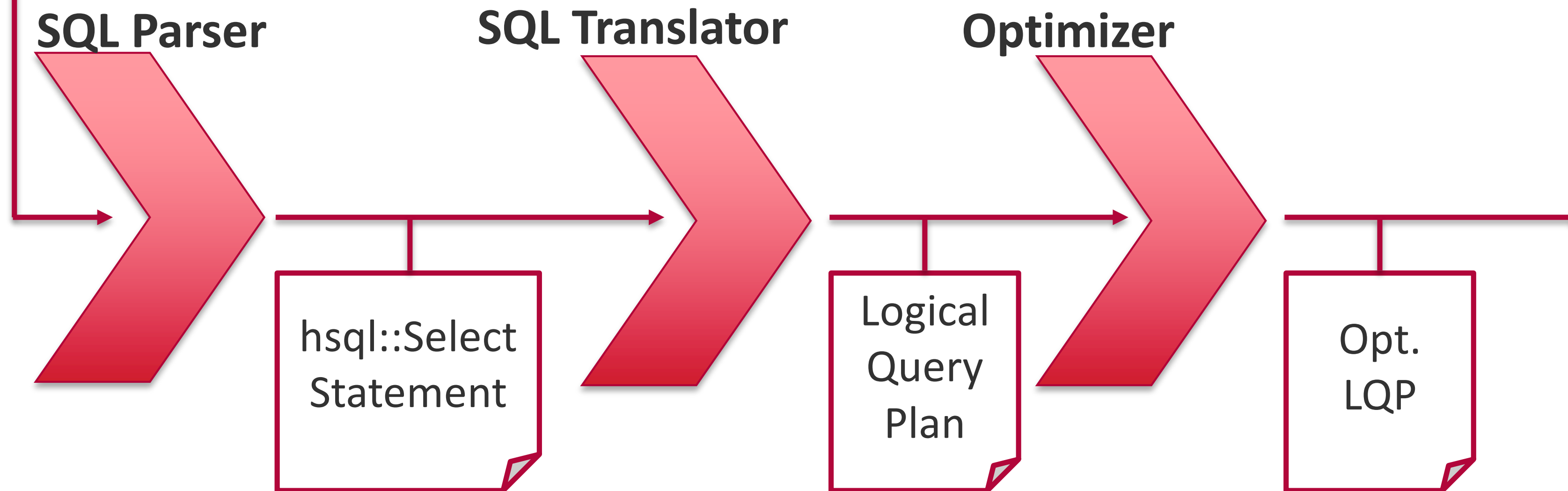
SQL Query

SELECT customer.id, COUNT(*) FROM customer LEFT JOIN order ON customer.id = order.cid WHERE order.priority = 1 GROUP BY customer.id



More Optimizer Rules

SQL Query
SELECT customer.id, COUNT(*) FROM customer LEFT JOIN order ON customer.id = order.cid WHERE order.priority = 1 GROUP BY customer.id



More Optimizer Rules

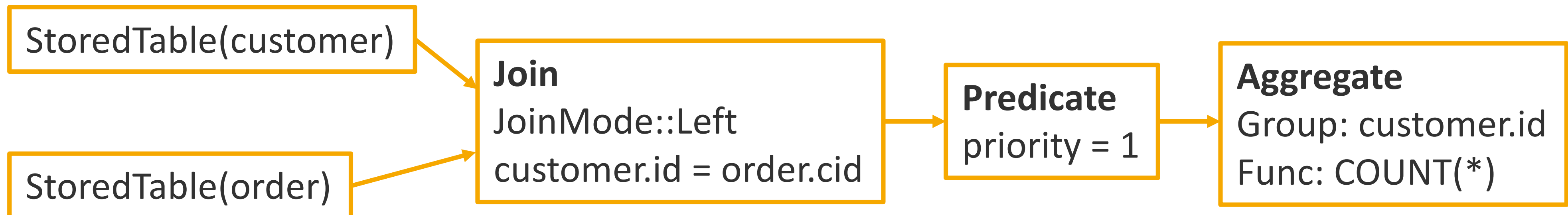
SQL Query
SELECT customer.id, COUNT(*) FROM customer LEFT JOIN order ON customer.id = order.cid WHERE order.priority = 1 GROUP BY customer.id

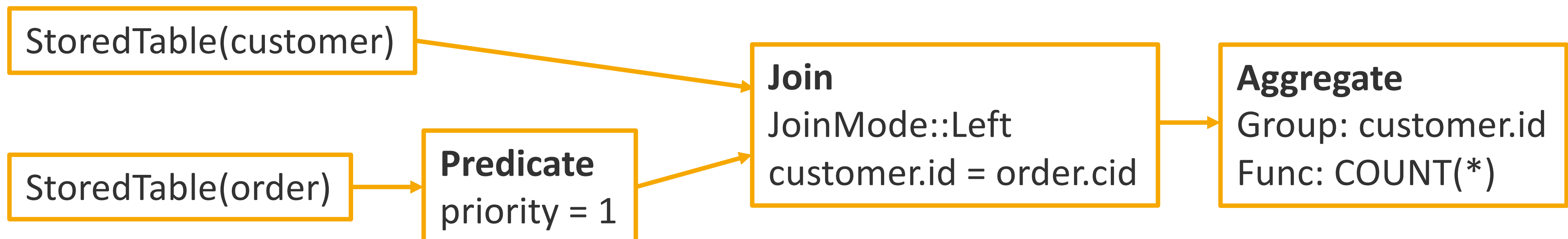
LogicalQueryPlan (LQP):

Nothing said about physical implementation of Join (HashJoin, NLJ, …)

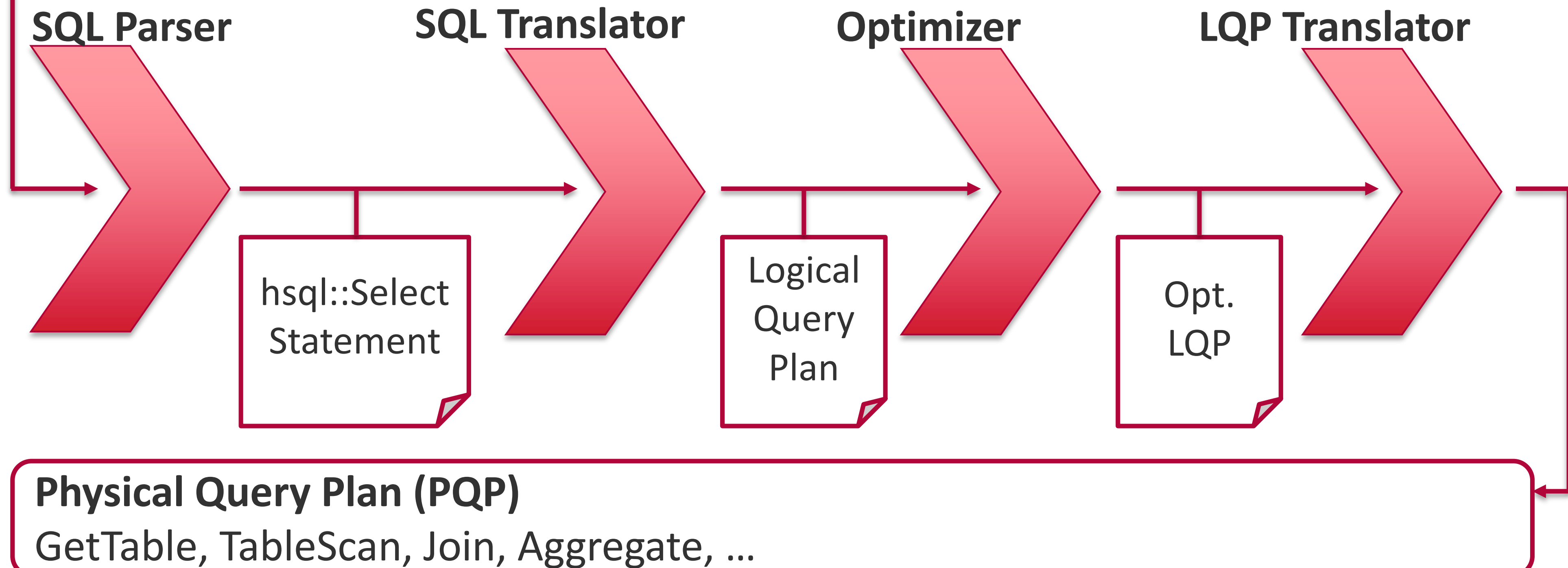
More Optimizer Rules

SQL Query
SELECT customer.id, COUNT(*) FROM customer LEFT JOIN order ON customer.id = order.cid WHERE order.priority = 1 GROUP BY customer.id





More Optimizer Rules



More Optimizer Rules

- Add more Rules to the Optimizer so that better decisions are made
- Example Rules:
  - Push-Down of Predicates
  - Arithmetic and Logical Optimizations
  - Making use of ordered output

Subqueries

Simple Subquery:

SELECT customer.id, COUNT(*) FROM customer LEFT JOIN **(SELECT * FROM order WHERE priority = 1)** ON customer.id = order.cid GROUP BY customer.id

Correlated Subquery:

SELECT customer.id, **(SELECT COUNT(*) FROM order WHERE customer.id = order.cid)** FROM customer GROUP BY customer.id

Subqueires

- Step 1: Extend the SQLTranslator to understand subqueries
  - Naïve version: Execute subquery for every row
- Step 2: Decorrelate Subqueries for better performance
  - E.g., replace them with a join

Topic Overview

- Joins
- Pruning Filters
- Networking
- Partitioning
- Strings, Date(times) and Functions
- Self-Driving Database
- More Optimizer Rules
- Subqueries

Logistics

- Together with your Sprint 3 Hand-In, please send us a list of **all topics that you are interested in**.
- All choices have the same priority and you can submit as many choices as you want.
- Next week, we will have a shorter lecture. After that, we will meet in the groups.

Scheduling

- Why scheduling?
  - Inter query and inter/intra operator parallelism
- Scheduler: Higher entity that offers interface for multiprocessing
- Implementation requirements:
  - Scheduling strategy should be exchangeable
  - Scheduler should be disableable
  - Simplicity of usage

Scheduling

- `AbstractTask` is base for schedulable units
- A `JobTask` is a general purpose task for anything that can parallelized
  - `JobTask(const std::function<void()>& fn)`
- Dependencies via `Task::set_as_predecessor_of`
  - Automatically notifies successors upon finish

Scheduling

```
auto jobs = std::vector<std::shared_ptr<AbstractTask>>{};

for (ChunkID chunk_id{0u}; chunk_id < chunk_count; ++chunk_id) {
  auto job_task = std::make_shared<JobTask>([=]() {
    // Actual scan goes here
  });

  jobs.push_back(job_task);
  job_task->schedule();
}

CurrentScheduler::wait_for_tasks(jobs);
```

HPI Hasso Plattner Institut

Motivation

- **A**tomicity
  - Transactions (Txs) either *commit* fully or not at all
- **C**onsistency
- **I**solation
  - Concurrent transactions must not affect each other
- **D**urability

Motivation – Textbook Example

Hasso Plattner Institut

CustomerName	Balance
John Smith	1,000.00 €
…	…

Transaction 1: John Smith pays 300 Euros for a flight

Tx 2: John Smith receives his 3000 Euro paycheck

Tx 1: UPDATE customers SET balance = balance – 300 …;

Tx 2: UPDATE customers SET balance = balance + 3000 …;

Tx 1: The DBMS reads the current balance

Tx 2: The DBMS reads the current balance

Tx 1: The DBMS calculates the new balance and writes it back

Tx 2: The DBMS calculates the new balance and writes it back

How much money does John have?

Types of Isolation Errors

- Lost Update
  - Two Tx modify the same row, but only one update is saved.
- Dirty Reads
  - A Tx reads changes that have not been committed yet.
- Non-Repeatable Reads
  - A Tx reads the same row twice and gets different results.
- Phantom Reads
  - A Tx sees rows that have been inserted during its lifetime.

Isolation Methods

- In general, databases have two ways to deal with concurrent modifications:
  - Pessimistic Two-Phase Locking (2PL)
  - Optimistic Concurrency Control (OCC)
- Hyrise and HANA use different flavors of OCC
  - Good if few conflicts are expected.

Multiversion Concurrency Control

- We do this by storing the commit id of the TX that added or deleted a row:

AccountId	Balance	BeginCid	EndCid
00119	1,000.00 €	0	5
00119	700.00 €	5	∞
00120	…		

- Additionally, we keep the transaction id to lock and mark transactions that are currently undergoing modifications.
  - We will not discuss this part right now.

Example with MVCC

Account	Balance	BeginCid	EndCid
X	7	5	∞
Y	0	5	∞

Global LastCid	5

Stores the commit id of the last Tx that committed, i.e., the timestamp of „now"

- We use the same example as for 2PL.
- Both rows were inserted by the last Tx, commit id 5.
- They are visible to future Txs because the BeginCid is equal and the EndCid is larger than the last commit id of new Txs.

For this lecture, we leave out some parts of MVCC (such as the two-phase commit) and discuss MVCC only on a conceptual level. For more details, please refer to the HYRISE documentation and Schwalb et al.: Efficient Transaction Processing for Hyrise in Mixed Workload Environments

Example with MVCC

Hasso Plattner Institut

Account	Balance	BeginCid	EndCid
X	7	5	∞
Y	0	5	∞

Global LastCid	5
Snapshot TxA	5

- Transaction A starts and retrieves the snapshot commit id from the global store.

Example with MVCC

Account	Balance	BeginCid	EndCid
X	7	5	∞
Y	0	5	∞

Global LastCid	5
Snapshot TxA	5

- It performs the check for sufficient funds in the account of X.

  Row 0 is visible because $5 \leq 5 < \infty$.

Example with MVCC

Account	Balance	BeginCid	EndCid
X	7	5	∞
Y	0	5	**6**
Y	5	6	∞

Global LastCid	**6**
Snapshot TxA	5

- 5 € are added to the account of Y. This happens by invalidating the old row and adding a new row.
- For simplicity, let us assume that TxA has commit id 6.

Example with MVCC

HPI Hasso Plattner Institut

Account	Balance	BeginCid	EndCid
X	7	5	**6**
Y	0	5	6
Y	5	6	∞
X	2	6	∞

Global LastCid	**6**
Snapshot TxA	5

- 5 € are subtracted from the account of X. Again, the old row is invalidated and the updated version of the row is appended.

Example with MVCC

HPI Hasso Plattner Institut

Account	Balance	BeginCid	EndCid
X	7	5	6
Y	0	5	6
Y	5	6	∞
X	2	6	∞

Global LastCid	6
TxA	completed
Snapshot TxB	6

- Transaction B starts. It retrieves the global LastCID 6.
- Row 0 cannot be used for validating, because its EndCid is not greater that TxB‘s snapshot commit id. It is invisible.
- Instead, row 3 is used for verification.

Multiversion Concurrency Control

- It is impossible to recover from conflicts.
- If a transaction finds that the row that it wants to invalidate has already been invalidated by another Tx, it has to abort and rollback all its changes.
- There is no guarantee that a Tx that started first will win this conflict. For highly contested rows, starvation is possible.

Next Steps

- Submit Sprint 3 and List of Preferred Projects
- Next Week
  - Review Sprint 3
  - Intro Hyrise Repo and Development Process
  - shared_ptr vs const shared_ptr& vs T&