SQLdays konferenz

2 Tage, 4 Tracks, 36 Sessions,
26 Speaker

„Can I join you“,

one table asked the other

Thomas Hütter
@DerFredo

Quest

Veranstalter

ppedv

“Can I join you“, one table asked the other

Thomas Hütter

“Can I join you?“, one table asked the other

Thomas Hütter, Diplom-Betriebswirt

- Application developer, consultant, accidental DBA, author
- Worked at consultancies, ISVs, end user companies
- SQL Server > 6.5, former „Navision“ > 3.0, R > 3.1.2
- Speaker at SQL events around Europe

@DerFredo https://twitter.com/DerFredo

de.linkedin.com/in/derfredo

www.xing.com/profile/Thomas_Huetter

SQLGLA sqlbits JOIN! THE DATABASE CONFERENCE SQLdays konferenz

Agenda

- Who was this Codd guy, what’s relational and why normalize?
- Our sample database
- Joins: CROSS, INNER, EQUI, NATURAL
- More Joins: OUTER to the LEFT and RIGHT, FULL, SELF, SEMI, ANTI
- Join’s best buddies Nested loop, Merge and Hash Joins, Adaptive
- Two more players: CROSS APPLY and OUTER APPLY
- Round-up; resources & credits; Q&A

Who was this Codd guy?

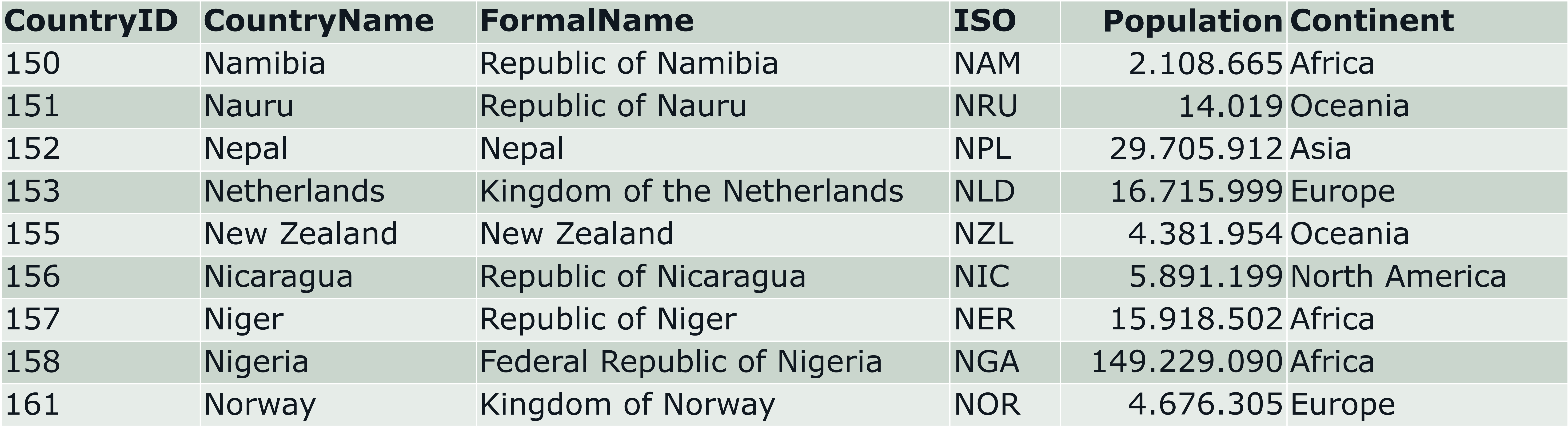
Edgar Frank „Ted“ Codd, PhD (1923 - 2003)

- English computer scientist, moved to US in 1948, working mainly for IBM
- was appointed IBM fellow, received the Turing award in 1981
- published „A relational model of data for large shared data banks“ in 1970 and Codd’s twelve rules (actually 13) in the mid 1980s,
  leading to relational DBs and the Structured query language, SQL
- Proposed Database normalization

The concept of a relational database

A relational database [management system] RDBMS

- organizes data in tables (relations) of rows (records or tuples) and columns (attributes), with (hopefully) a unique key identifying each row



CountryID	CountryName	FormalName	ISO	Population	Continent
150	Namibia	Republic of Namibia	NAM	2.108.665	Africa
151	Nauru	Republic of Nauru	NRU	14.019	Oceania
152	Nepal	Nepal	NPL	29.705.912	Asia
153	Netherlands	Kingdom of the Netherlands	NLD	16.715.999	Europe
155	New Zealand	New Zealand	NZL	4.381.954	Oceania
156	Nicaragua	Republic of Nicaragua	NIC	5.891.199	North America
157	Niger	Republic of Niger	NER	15.918.502	Africa
158	Nigeria	Federal Republic of Nigeria	NGA	149.229.090	Africa
161	Norway	Kingdom of Norway	NOR	4.676.305	Europe

The concept of a relational database

A relational database [management system] RDBMS

- organizes data in tables (relations) of rows (records or tuples) and columns (attributes), with (hopefully) a unique key identifying each row
- provides relational operators to manipulate the data UNION, INTERSECT, EXCEPT, JOIN
- supports ACID transactions (Atomicity, Consistency, Isolation, Durability)
- follows certain rules of normalization in order to prevent redundancy, data manipulation anomalies and loss of data integrity
- Top 5 *): Oracle, MySQL, MS SQL Server, PostgreSQL, IBM DB2

*: „popularity“ according to db-engines.com, October 2019

Database normalization / normal forms

- Unnormalized form, UNF
  Can contain redundant data, risk of anomalies after CRUD *) operations
- First normal form, 1NF
  Unique primary keys, no repeating groups
- Second normal form, 2NF
  1NF + no non-prime attribute is dependent on any part of the key
- Third normal form, 3NF
  2NF + no non-prime attributes depends on any other non-prime attribute
  „Every non-key attribute must provide a fact about the key, the whole key, and nothing but the key, so help me Codd.“ 😀 [Bill Kent, George Diehr]

*: Create, Read, Update, Delete

Our sample database

Wide World Importers

- The „new“ MS sample database for the core database features of SQL Server from version 2016 and Azure SQL Database
- Follow-up to Northwind and AdventureWorks DBs
- Can be downloaded from a GitHub repository, plus there are additional scripts, Visual Studio solutions, etc…
- Represents a „wholesale novelty goods importer and distributor“
- Contains ca. 663 customer records, 70000 sales invoices, > 200000 warehouse movements
- Demo: denormalized SalesInvoices table

Joins

- CROSS
  returns the Cartesian product of rows in both tables, that is each row from the first table combined with each row in the second table

  13 rows in table_A * 9 rows in table_B -> 117 rows result set

  explicit:
  ```
  SELECT * FROM table_A CROSS JOIN table_B
  ```

  implicit:
  ```
  SELECT * FROM table_A, table_B
  ```

Joins

- INNER

  combines rows from two tables based on matching column values determined by the join predicate(s)

  explicit:

  ```
  SELECT * FROM table_A [INNER] JOIN table_B
  ON table_B.column1 = table_A.column3
  ```

  implicit:

  ```
  SELECT * FROM table_A, table_B
  WHERE table_B.column1 = table_A.column3
  ```

Joins

- EQUI
  the join only uses equality comparisons (=) in the predicates

  shorthand form if column names equal (not in SQL Server ☹️)

  ```
  SELECT * FROM table_A JOIN table_B USING (column_x)
  ```

- NATURAL (also not in SQL Server)
  combines tables by equality on their common column names

  ```
  SELECT * FROM table_A NATURAL JOIN table_B
  ```

More Joins

- LEFT [OUTER]

  returns all rows from the left table

  - combined with column values of matching rows from the right table

  - right columns contain NULL values if no match found

  ```
  SELECT * FROM table_A
  LEFT [OUTER] JOIN table_B
  ON table_B.column1 = table_A.column1
  ```

More Joins

- RIGHT [OUTER]
  returns all rows from the right table
  - combined with column values of matching rows from the left table
  - left columns contain NULL values if no match found

```
SELECT * FROM table_A
RIGHT [OUTER] JOIN table_B
ON table_B.column1 = table_A.column1
```

More Joins

- FULL [OUTER]
  combines applying a LEFT and a RIGHT OUTER JOIN:
  one row for each match between two tables
  + rows from left table not matched in right
  + rows from right table not matched in left

```
SELECT * FROM table_A
FULL [OUTER] JOIN table_B
ON table_B.column1 = table_A.column1
```

More Joins

- SELF
  joining one table not to another table, but to itself

```
SELECT * FROM table_A A1
INNER JOIN table_A A2
ON A2.column1 = A1.column1
WHERE A1.PK < A2.PK
```

More Joins

- SEMI

  returns all rows (each only once) of the first table for which there is a match in the second table

  direct syntax (not in SQL Server)

  ```
  SELECT * FROM table_A SEMI JOIN table_B
  ON table_B.column1 = table_A.column1
  ```

  indirect syntax using `EXISTS`, `IN` or `INTERSECT`

More Joins

- ANTI

  returns all rows of the first table for which there is *no* match in the second table

  direct syntax (not in SQL Server)

  ```
  SELECT * FROM table_A ANTI JOIN table_B
  ON table_B.column1 = table_A.column1
  ```

  indirect syntax using `NOT EXISTS`, `NOT IN` or `EXCEPT`

Nested loop, merge and hash joins

- Nested loops join

  If one table is rather small and the other fairly large and indexed on its join column(s), a nested loop is the fastest join, requiring minimum I/O operations and comparisons.

  The outer loop consumes the outer table row-by-row, and for each outer row, searches for matches in the inner table.

  Naive n l j: scans the entire table or index
  Index(ed) n l j: exploits an index
  Temporary index n l j: builds + destroys index as part of the query plan

Nested loop, merge and hash joins

- Merge join
  If the two inputs are fairly large, similarly sized and are sorted on their join column, a merge join often is the fastest option.

  Basically, the merge join operation fetches one row from each input and compares them. In the case of an equi join, they are returned if equal.
  If they are not equal, the lower valued row is ignored and the next row is fetched from that input. This is repeated until all rows are processed.

  Many-to-many may require use of temporary tables, duplicate values may require rewinds. On-the-fly sorting may occur.

Nested loop, merge and hash joins

- Hash join
  May be the best choice for large inputs that are unsorted and not indexed.

  A hash table is generated in memory, based on the *build* input. Then the *probe* input is scanned, hash calculated and matched.

  In memory h j: the whole build input fits into memory.
  Grace h j: build and probe in several steps/phases.
  Recursive h j: for XL build inputs, multiple partitioning steps/levels and async I/O.

  May start in memory, then dynamically change to grace, then recursive.
  Also, build and probe may be reversed („role reversal“) if wrongly estimated.

Adaptive joins

- Adaptive/Intelligent query processing features

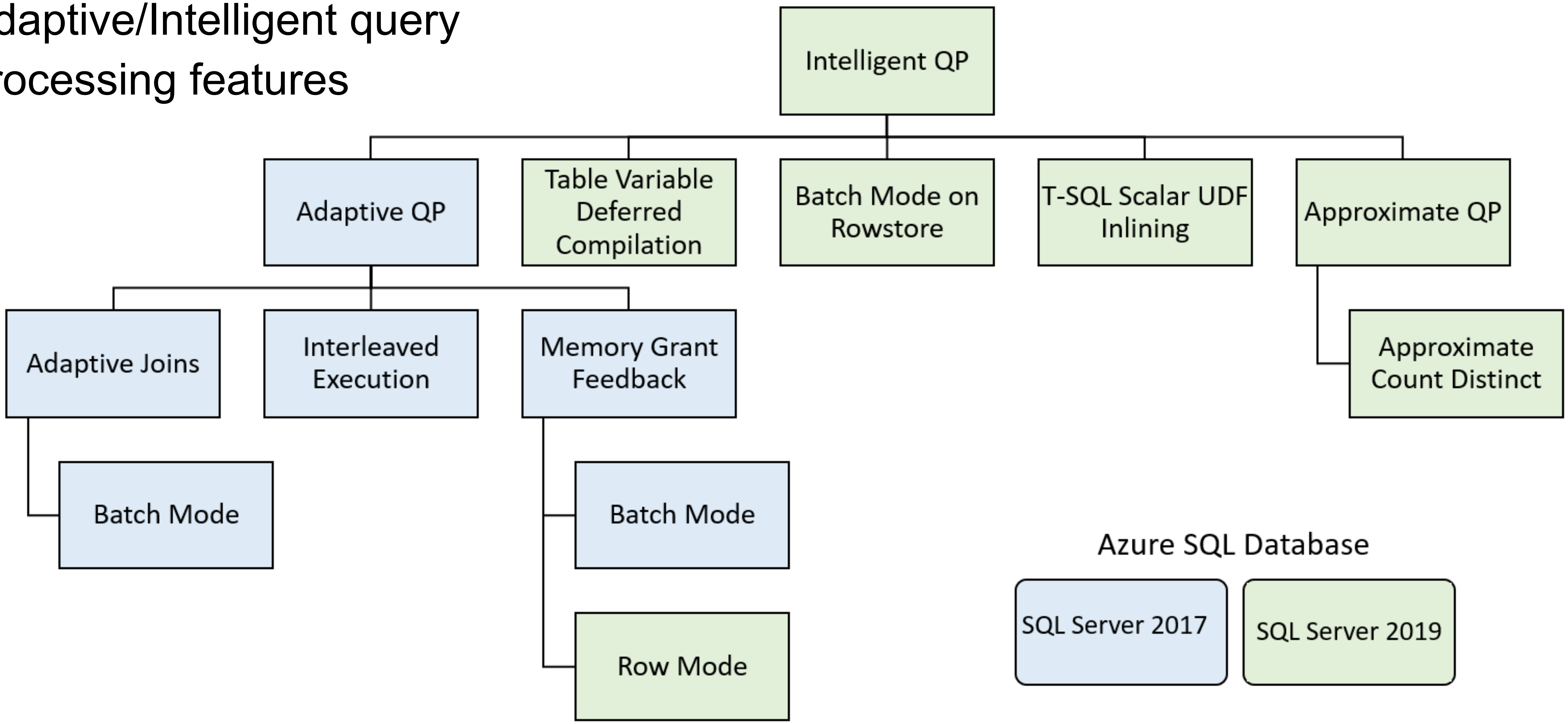


Illustration © Microsoft https://docs.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing?view=sql-server-2017

Adaptive joins

- (Batch mode) Adaptive joins
  - Were introduced in SQL Server 2017 as one of the „Adaptive Query Processing“ features.
  - High level explanation:
    for large result sets, often a hash join is faster than a nested loop join.
  - The decision between hash and nested loop join
    to be made „on the fly“, depending on input data.
  - Threshold is determined based on estimated number of rows / cost.
  - Works for batch mode only (Columnstore index involved).
  - Hint: JOIN type can be forced by a hint (know what you’re doing!)

Cross Apply and Outer Apply

- Used when right part contains a table-valued expression or aggregate
- In some cases, using APPLY increases query performance significantly

- CROSS APPLY
  is equivalent to a CROSS JOIN

- OUTER APPLY
  is equivalent to a LEFT OUTER JOIN

Round-up

- Ted Codd, relational databases and SQL, normalization
- Wide World Importers sample database
- JOINs, logical perspective: CROSS, INNER, LEFT, RIGHT…
- JOINs, technical perspective: loop, merge, hash, adaptive, (hints)
- Cross and Outer Apply
- SQL Server on Docker, Azure Data Studio,

  Jupiter (SQL) Notebooks, Query plans

Resources on- and offline, credits

- „Ted“ Codd: https://en.wikipedia.org/wiki/Edgar_F._Codd
  Relational database: https://en.wikipedia.org/wiki/Relational_database
  Database normalization: https://en.wikipedia.org/wiki/Database_normalization
- Ranking of RDBMS: https://db-engines.com/en/ranking/relational+dbms
- Joins in ANSI-SQL: https://en.wikipedia.org/wiki/Join_(SQL)
  Joins in MS SQL Server
  - logical view:https://docs.microsoft.com/en-US/sql/t-sql/queries/from-transact-sql#join-type
  - technical view: https://docs.microsoft.com/en-US/sql/relational-databases/performance/joins
  - semi and more: https://sqlperformance.com/2018/02/sql-plan/row-goals-part-2-semi-joins
  - query hints: https://docs.microsoft.com/en-US/sql/t-sql/queries/hints-transact-sql
- Wide World Importers, Microsoft’s current sample database
  - to play along, download the WideWorldImporters-Full.bak file from:
  https://github.com/Microsoft/sql-server-samples/releases/tag/wide-world-importers-v1.0
  - repo including code samples: https://github.com/microsoft/sql-server-samples

“Can I join you?‟, one table asked the other

Time for some Q & A?

And here’s your first answer: 😉

Yes, this file and the demo script can be found at:
https://github.com/SQLThomas/Conferences/tree/master/Erding2019

“Can I join you?‟, one table asked the other

Thank you for your time and interest & keep in touch:

@DerFredo https://twitter.com/DerFredo

de.linkedin.com/in/derfredo

www.xing.com/profile/Thomas_Huetter