HOW TO ACCESS AND LINK DATA USING ENTITY/RELATIONSHIP PROFILING

Whitepaper

SUMMARY

In this paper, we will define what linked data is within the context of an organization, and how to use Entity/Relationship (E/R) Profiling to link your data. Using the DataViadotto Profiler, we showcase the process on a publicly available Hockey database. Despite the fact that the database and its schema have been curated for decades, our E/R Profiler is capable of finding new sensible keys and new sensible foreign keys that optimize data access and link the data in more meaningful ways. Interestingly, we also find violations of referential integrity arising from sensible foreign keys that have never been specified before we discovered them with our profiler.

1. Linked Data Revolutions

Sir Tim Berners-Lee's great vision of the Semantic Web depends crucially on *Linked Data*, which crucially depends on Uniform Resource Identifiers (URIs), a global id that uniquely identifies resources on the Web and elsewhere. The vision is neat and ambitious. If implemented, information can be shared easily and efficiently. However, it remains to be seen whether people will put up time and work to provide sufficient meta data to put the vision into reality.

Instead of aiming for linked data globally, a first step would be to aim for linked data locally, such as your organization, a branch of it, perhaps just within a single data project. In fact, do you think that even tables of a, supposedly, well-designed database are linked, or even just accessible?

In practice, many database people do use *surrogate identifiers* as keys, and other tables may reference such an identifier via so-called foreign keys. Such key / foreign key relationships form the foundation of Ted Codd's relational model of data. They are fundamental to entity and referential integrity; and implement Peter Chen's Entity/Relationship Model.

However, ids that are artificially created, such as auto-increments, carry no actual meaning and should therefore be hidden from the user. They are meant for internal use to facilitate efficient data processing. In fact, surrogate ids must never *replace* natural (real-world) keys, which are combinations of table fields that uniquely identify every record of the table. Hence, they provide efficient access to all business entities in your tables. If no natural key is specified, there is no mechnism that prevents the duplication of business entities. For example, the same customer may have multiple ids. It is not difficult to imagine how bad reporting and analytics may become in such cases. However, the use of one natural key cannot only avoid the assignment of multiple ids to the same business entity, but even detect which ids have been incorrectly assigned to the same business entity by grouping business entities according to the

values on the natural key. So, if you use a surrogate identifier, you ought to use some natural key. In fact, you need to specify every natural key that represents a business rule. There are multiple reasons for that. 1) If you do not specify any natural key, you have no means to prevent records with duplicate values on all the fields of this key. In other words, you permit the duplication of entities. 2) You miss an opportunity to identify business entities. 3) You miss an opportunity to efficiently access your entities based on a UNIQUE index, which will make update and query operations faster when specified. 4) You miss an opportunity to provide a point of reference in the form of a foreign key/key relationship, which will allow you to efficiently join this table with others. Indeed, we cannot overemphasize the benefit for making database tables accessible for other data sets, internal or external to your project. Indeed, tables with foreign keys that reference surrogate ids imply that the surrogate id must already be present in the tables. Of course, this is no problem within the context of a database model, but it is a problem when the database needs to be integrated or linked with other data. Other aspects include the ability to understand the results of queries and reports, and to execute them efficiently. The use of surrogate ids in query answers makes them difficult to understand, and adding fields to the results to make them more comprehensible may mean that additional, potentially expensive, join operations become necessary.

As a conclusion, we define **linked data** as tables linked by foreign keys that reference natural keys. Importantly, linked data are open for access, comprehension, discovery and integration. This is particularly important within the era of data, where companies have no choice but to assess the relevance of any data set for their projects, and to integrate such data sets within their data repository. Only this way, business entities and their relationships can be discovered and assembled into data stories, providing insight and benefit to the organization and their customers.

2. The Hockey Database

The Hockey data set is publicly accessible at https://relational.fit.cvut.cz/dataset/Hockey and was sourced from http://www.opensourcesports.com/hockey/. In addition to the NHL, the Hockey data set covers also early and alternative leagues: NHA, PCHA, WCHL and WHA. It contains individual and team statistics from the 1909/10 through to the 2011/12 season. Together, it contains 22 tables, 96,403 rows and 300 columns, and has a size of 15.6 MB.

The original conceptual data model is illustrated in Fig. 1 on the next page. Out of the 22 tables, nine tables have neither a primary key nor any unique constraints specified on them, while the remaining 13 tables have only a primary key specified on them without any other unique constraint. When a field name is part of the primary key of a table, the name of the field is underlined and the letters PK for Primary Key appear next to it. Some of the referential integrity constraints are not foreign keys since they do not reference a unique constraint (this is a minimal requirement on any foreign key, and it means, in particular, that those constraints do also not reference the primary key of the table if it exists). Not having a canidate key specified on the table and having referential constraints that are not foreign keys violates basic design principles. There are other database design issues, such as providing a single table (the table called Master) for different people such as players, coaches, managers etc., which is one of the reasons why no candidate key exists for this table.

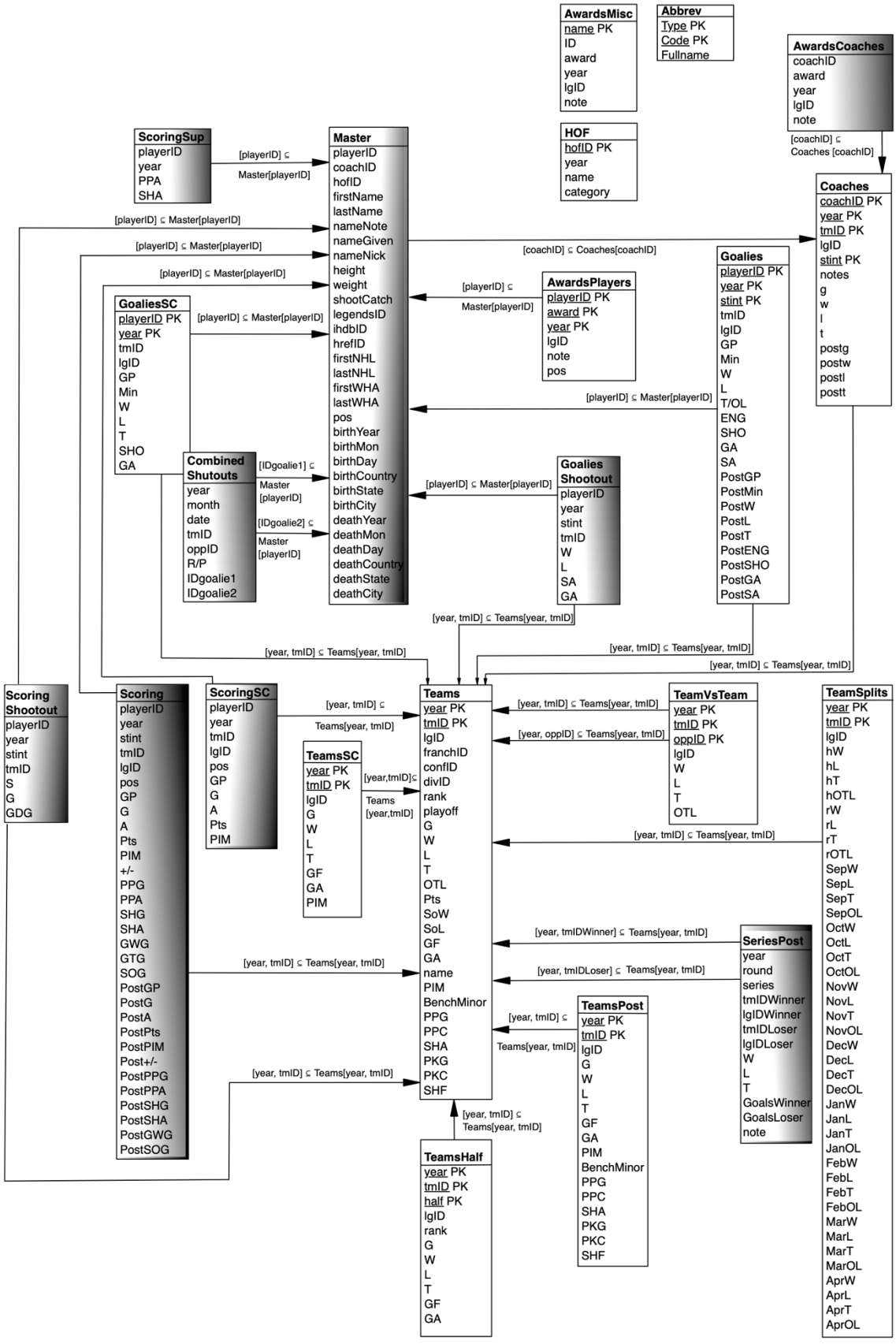


Fig. 1: Details of the Original Conceptual Diagram for the Hockey Data Set

3. Making Database Tables Accessible

As an illustration how E/R Profiling can benefit data access and linkage, we will examine a small snippet of the Hockey database. Two of the tables for which no primary key nor uniqueness constraint have been specified are *Scoring* and *ScoringShootout*, respectively. The former table records various statistics how players scored goals in games over a season. If a game in a regular season is tied at the end of the five-minute overtime, the game goes to a shootout, with each team given three "penalty shots", to determine the winner. The interested reader may find information on the various hockey statistics in Wikipedia.

It is a very interesting and important question to ask how records over these two tables can be accessed efficiently. Surprisingly, no primary key nor uniqueness constraints have been specified.

The DataViadotto Key Finder returns, within 1 second, one minimal candidate key and two minimal uniqueness constraints for Scoring, and two minimal candidate keys for ScoringShootout when we look for such constraints on those tables with up to four fields. These are shown in Fig. 2.

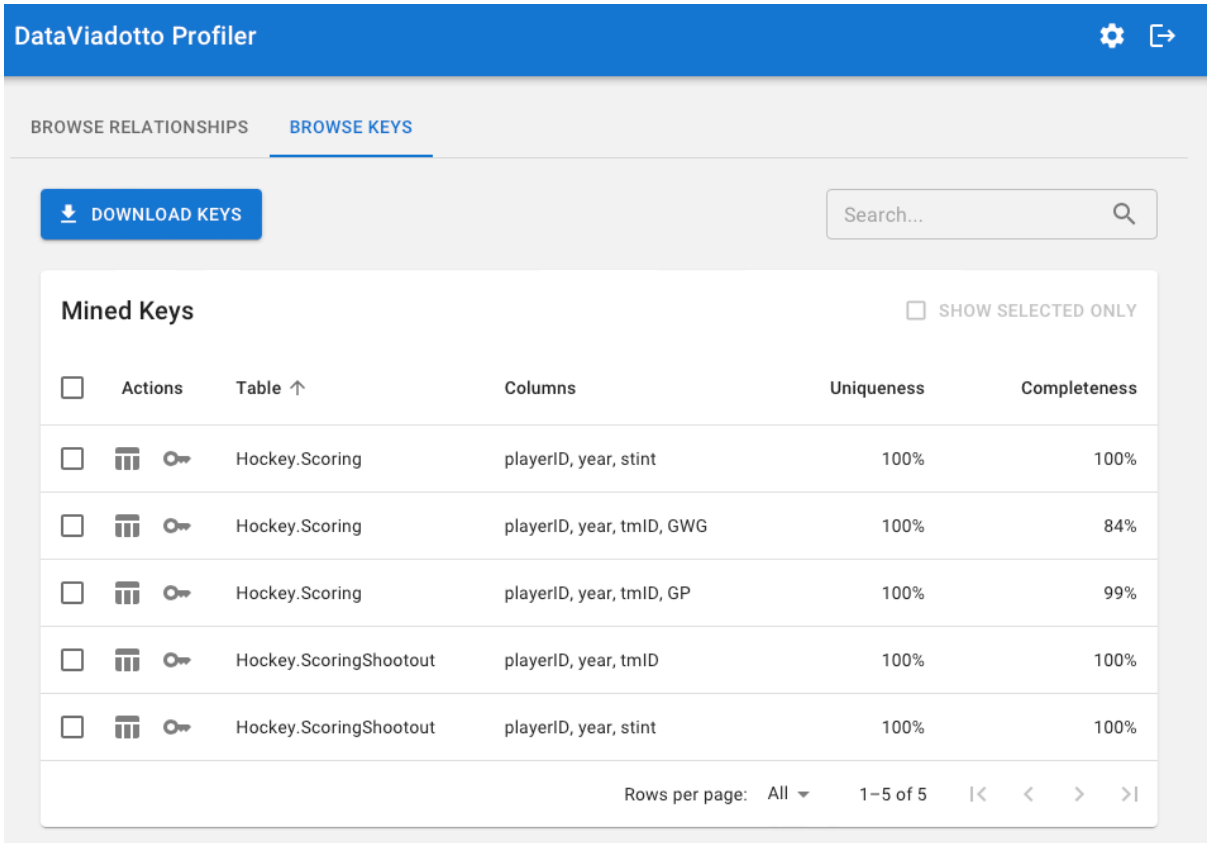
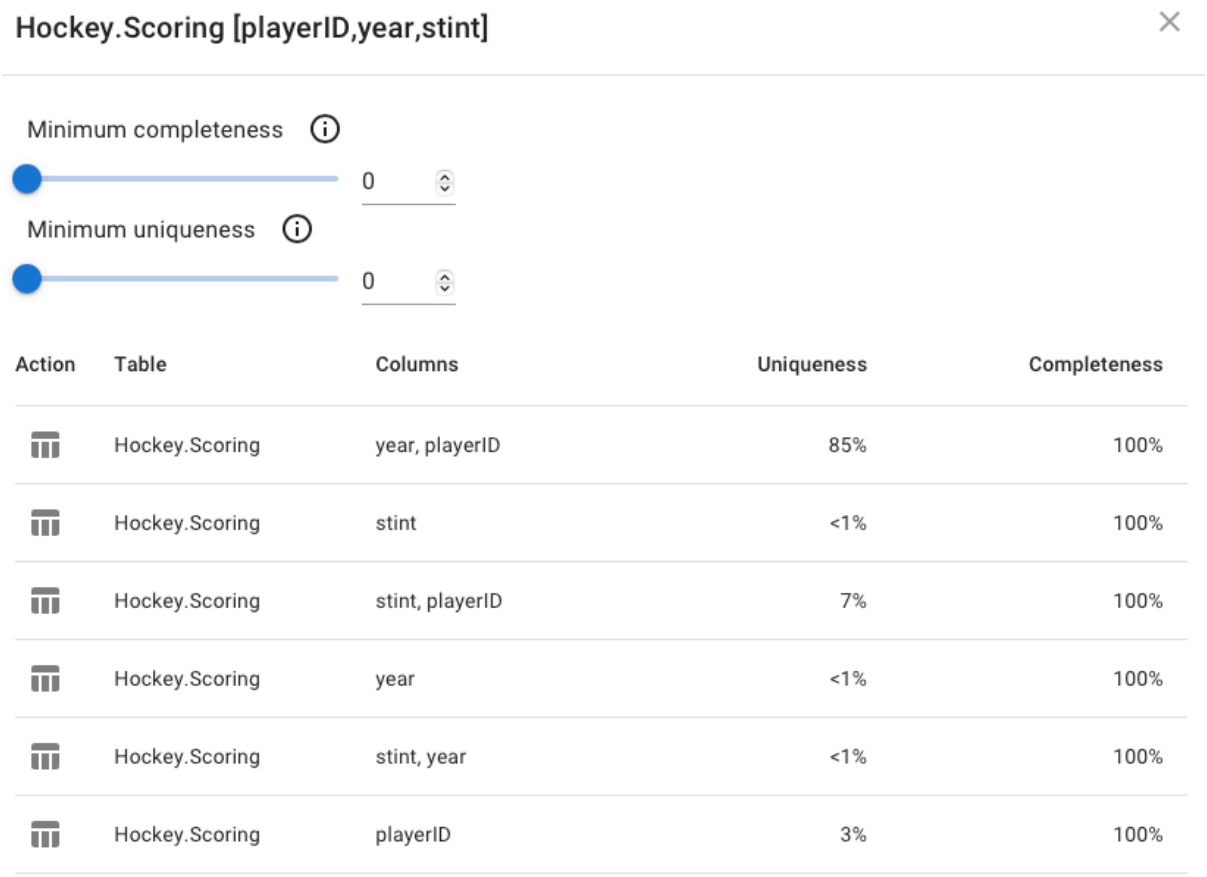


Fig. 2: Minimal Candidate Keys and Uniqueness Constraints on Scoring and ScoringShootout Mined

For table *Scoring*, the only candidate key is *{playerID, year, stint}*, which is minimal, meaning that removal of any field from {playerID, year, stint} is not a candidate key.

Indeed, a *subset-key analysis* is shown in Fig.3. Here, all proper and non-empty subsets of the key *{playerID, year, stint}* are shown together with their uniqueness ratio (the percentage of rows uniquely identified by values in columns of the subset or containing null on some column of the subset) and completeness ratio (percentage of rows that have no null on any column of the subset). In particular, *{playerID, year}* only has a uniqueness ratio of 85%.



Action	Table	Columns	Uniqueness	Completeness
	Hockey.Scoring	year, playerID	85%	100%
	Hockey.Scoring	stint	<1%	100%
	Hockey.Scoring	stint, playerID	7%	100%
	Hockey.Scoring	year	<1%	100%
	Hockey.Scoring	stint, year	<1%	100%
	Hockey.Scoring	playerID	3%	100%

Fig. 3: Results of Subset-Key Analysis for {playerID, year, stint}

An inspection of a data sample for *{playerID, year}* brings up several combinations where the same player has played for different teams in the same year (but at different stints). A snippet of such data sample is shown in Fig. 4, and the duplicate records (pairs of records with matching values on *playerID* and *year*) have a background in red.

We conclude that *{playerID, year, stint}* forms a sensible, minimal candidate key that should be specified on table *Scoring*.

Hockey.Scoring [year,playerID]

playerID	year	stint	tmID	lgID	pos	GP	G	A	Pts	PIM	+/-	PPG
alleyst01	1978	1	BIR	WHA	L	78	17	24	41	36	-5	4
alleyst01	1980	1	HAR	NHL	L	8	2	2	4	11	1	0
beverbi01	1930	1	OTS	NHL	G	9	0	0	0	0	null	null
romnedo01	1930	1	CHI	NHL	L/C	30	5	7	12	8	null	null
nokelpe01	2009	1	AND	NHL	C	50	4	7	11	21	-7	0
nokelpe01	2009	2	PHO	NHL	C	17	1	1	2	6	-2	0
thompna01	2009	1	NYI	NHL	C	39	1	5	6	39	-14	0
thompna01	2009	2	TBL	NHL	C	32	1	3	4	17	-3	0
torrera01	2009	1	CBS	NHL	L	60	19	12	31	32	-8	7
torrera01	2009	2	BUF	NHL	L	14	0	5	5	2	-3	0

Fig. 4: Example data illustrating that the same player may have played for different teams in the same year

Fig. 2 also showed that the uniqueness constraints *{playerID, year, tmID, GP}* and *{playerID, year, tmID, GWG}* hold with 99% and 84% completeness ratios, respectively. Here, *GP* stands for "Games Played" and *GWG* stands for "Game-Winning Goals". Since the two UCs are minimal, their common subset *{playerID, year, tmID}* cannot have a uniqueness ratio of 100%. Indeed, the ratio is 99%, so perhaps this is an indication that duplicate entities are present? However, inspecting data examples of Fig. 5 for *{playerID, year, tmID}* reveals that the same player may have played for the same team in the same year at different stints. Of course, this happens rarely, as it requires a player to move to a different team for stint 2 and return to the original team from stint 1 for stint 3, so they played for the same team in stint 1 and 3 of the same year. As this can happen, *{playerID, year, tmID}* is not a sensible key.

In addition, the UCs *{playerID, year, tmID, GP}* and *{playerID, year, tmID, GWG}* are also not sensible. It just means it has not happened so far that a player who played for the same team in the same year at different stints has played the same number of games for these two stints, or scored the same number of game-winning goals for these two stints. Hence, these uniqueness constraints should not be specified since this may well happen in the future.

Hockey.Scoring [tmID,year,playerID]

playerID	year	stint	tmID	lgID	pos	GP	G	A	Pts	PIM	+/-	PPG	PPA	SHG	SHA	GWG
heylivi01	1937	1	CHI	NHL	C	7	0	0	0	0	null	null	null	null	null	null
ruzicst01	2006	1	PHI	NHL	R	40	3	10	13	18	-6	1	3	0	0	0
bruceda01	1988	1	VAN	NHL	R	53	7	7	14	65	-16	1	3	0	0	2
lerouje01	1997	1	CHI	NHL	L	66	6	7	13	55	-2	0	0	0	0	0
lorimbo01	1984	1	NJD	NHL	D	46	2	6	8	35	6	0	null	0	null	0
laperja01	1972	1	MTL	NHL	D	57	7	16	23	34	78	2	null	0	null	0
mattsma01	1977	1	WIJ	WHA	G	6	0	0	0	0	0	0	null	0	null	null
mattsma01	1977	3	WIJ	WHA	G	4	0	0	0	0	0	0	null	0	null	null
kennede01	1988	1	LAK	NHL	D	25	2	5	7	23	14	0	0	0	null	1
kennede01	1988	3	LAK	NHL	D	26	1	3	4	40	4	0	0	0	null	0

Fig. 5: Data examples showing players who returned to their original team from stint 1 for stint 3 (while playing for a different team for stint 2)

We now turn to an analysis of the uniqueness constraints mined from the table *ScoringShootout*. Indeed, Fig. 2 shows the two minimal candidate keys *{playerID, year, tmID}* and *{playerID, year, stint}* as the only results. The first one may not be sensible, since it would prevent us from recording players that participate in shootouts of games for the same team during stint 1 and stint 3 of the same year (on return after playing for a different team in stint 2). However, the second key *{playerID, year, stint}* is indeed sensible. In fact, the same player can only play for one team during the same stint in the same year. This would only require one record in table *ScoringShootout*.

As the conclusion to this chapter, Entity/Relationship Profiling has brought forward sensible primary key candidates for each table *Scoring* and *ScoringShootout*, which had not been specified after decades of use. After specifying these keys, records in those tables becomes efficiently accessible in time in *O(log n)* where *n* denotes the number of records in these tables, using a B-tree *UNIQUE* index. As we will see in the next chapter, specifying such keys has the additional advantage of making the tables linkable.

4. Linking Database Tables

Among other relationships, Fig. 1 illustrated that table ScoringShootout is linked to table Team using the foreign key *[tmID,year]* $\subseteq$ *Teams[tmID,year]*, that is, for every record over ScoringShootout there is a unique record over table *Teams* with matching values on *tmID* and *year*. In other words, for every player participating in a shootout, the team and year in which he plays for this team must refer to a unique team recorded in the table Teams for that year. As illustrated in Fig. 6, the DataViadotto Profiler discovers this foreign key easily.

Mined Relationships

SHOW SELECTED ONLY

Action	Source table	Target table	Source columns	Target columns	Inclusion (simple)	Inclusion (partial)	Inclusion (full)	Coverage	Max cardinality	Uniqueness	Join type
	Hockey.ScoringShootout	Hockey.Teams	year, tmID	year, tmID	100%	100%	100%	13%	17	100%	

Rows per page: All 1–1 of 1

Fig. 6: Existing Foreign Key from ScoringShootout to Teams

Fig. 7 shows example data that validates the foreign key.

Hockey.ScoringShootout [year,tmID] -> Hockey.Teams [year,tmID]

Hockey.ScoringShootout

playerID	year	stint	tmID	S	G	GDG
coglian01	2007	1	EDM	1	0	0
gagnesa01	2007	1	EDM	17	5	4
boothda01	2007	1	FLO	1	0	0
bouwmja01	2007	1	FLO	1	0	0
browndu01	2008	1	LAK	6	1	0
doughdr01	2008	1	LAK	3	1	1
belaner01	2007	1	MIN	1	0	0
bouchpi02	2007	1	MIN	3	1	0
backlmi01	2011	1	CAL	0	1	0
bourqre01	2011	1	CAL	0	1	0

Hockey.Teams

year	lgID	tmID	franchID	confID	divID	rank	playoff	G	W
2007	NHL	EDM	EDM	WC	NW	4	null	82	41
2007	NHL	FLO	FLO	EC	SE	3	null	82	38
2008	NHL	LAK	LAK	WC	PC	5	null	82	34
2007	NHL	MIN	MIN	WC	NW	1	CQF	82	44
2011	NHL	CAL	CAL	WC	NW	2	null	82	37
2005	NHL	STL	STL	WC	CE	5	null	82	21
1991	NHL	BUF	BUF	WA	AD	3	DSF	80	31

Fig. 7: Example data validating the foreign key from ScoringShootout to Teams

However, having just identified *{playerID, year, stint}* as sensible key for both tables *ScoringShootout* and *Scoring,* we can specify the foreign key

$$[playerID, year, stint] \subseteq Scoring[playerID, year, stint].$$

Indeed, as Fig. 8 demonstrates, this is a one-to-one relationship (also called a specialisation) and requires a right-outer join between *ScoringShootout* and *Scoring*. We have just linked tables that were previously unlinked.

Action	Source table	Target table	Source columns	Target columns	Inclusion (simple)	Inclusion (partial)	Inclusion (full)	Coverage	Max cardinality	Uniqueness	Join type
	Hockey.ScoringShootout	Hockey.Scoring	playerID, year, stint	playerID, year, stint	100%	100%	100%	5%	1	100%	

Fig. 8: New foreign key from ScoringShootout to Scoring

There are further interesting benefits. Indeed, the foreign key *ScoringShooutout[tmID,year] ⊆ Teams[tmID,year]* has become redundant (and is therefore not needed) since we now have the two foreign keys *ScoringShootout[playerID, year, stint] ⊆ Scoring[playerID, year, stint]* and *Scoring[tmID, year] ⊆ Teams[tmID, year]*, and the fact that value on *tmID* is uniquely determined by the value combination on *playerID*, *year*, and *stint* (every player can only play for one team in every year during every stint). Moreover, the field *tmID* is not required on the

table *ScoringShootout*. It is redundantly repeated on that table due to the foreign key *ScoringShootout[playerID, year, stint] ⊆ Scoring[playerID, year, stint]*.
The redundancy of field *tmID* on *ScoringShootout* has resulted in data inconsistency. The *ScoringShootout* table mentions the player "Kevyn Adams" played in the 2006 season for the "Phoenix Coyote" in stint 1, but the *Scoring* table says the same player played for the "Carolina Hurricanes" in 2006 in stint 1, and for the "Phoenix Coyote" in Stint 2. This inconsistency leaves open for which team "Kevyn Adams" scored in a shootout that year. If we leave the *tmID* in *ScoringShootout* aside (as it is redundant anyway), then "Kevyn Adams" scored in a shootout in stint 1 in 2006, and he did that for the "Carolina Hurricanes". This inconsistency occurs since the correct foreign key *ScoringShootout[playerID, year, stint] ⊆ Scoring[playerID, year, stint]* has never been specified. In fact, the inconsistency above can be discovered as a result of mining the inclusion dependency:

$$ScoringShootout[playerID, year, stint] \subseteq Scoring[playerID, year, stint]$$

which only holds with 99% inclusion ratio, as illustrated in Fig. 9.

Action	Source table	Target table	Source columns	Target columns	Inclusion (simple)	Inclusion (partial)	Inclusion (full)	Coverage	Max cardinality	Uniqueness	Join type
	Hockey.ScoringShootout	Hockey.Scoring	playerID, year, stint, tmID	playerID, year, stint, tmID	99%	99%	99%	5%	1	100%	1 1

Fig. 9: Inclusion dependency from ScoringShootout to Scoring with ratios of 99%

Fig. 10 illustrates how a data example identifies the data inconsistency above. The playerID "adamske01" is for player "Kevyn Adams", and "Pho" is the tmID for team "Phoenix Coyote".

Hockey.ScoringShootout [playerID,year,stint,tmID] -> Hockey.Scoring [playerID,year,stint,tmID]

playerID	year	stint	tmID	S	G	GDG
malonry01	2009	1	TBL	1	0	0
bradlma01	2006	1	WAS	1	0	0
fiddlve01	2008	1	NAS	6	2	0
carteje01	2006	1	PHI	2	0	0
ribeimi01	2009	1	DAL	12	1	0
smithcr01	2011	1	NAS	0	4	0
sextoda01	2009	1	AND	2	0	0
yandlke01	2009	1	PHO	1	0	0
armstde01	2006	1	LAK	1	0	0
adamske01	2006	1	PHO	1	0	0

Hockey.ScoringShootout

Hockey.Scoring

playerID	year	stint	tmID	lgID	pos	GP	G	A
brouwtr01	2010	1	CHI	NHL	R	79	17	1
nealja01	2008	1	DAL	NHL	L	77	24	1
pitkajo01	2006	1	PHI	NHL	D	77	4	3
morribr01	2010	1	CAL	NHL	C	66	9	3
malonry01	2009	1	TBL	NHL	L	69	21	2
bradlma01	2006	1	WAS	NHL	R	57	4	9
fiddlve01	2008	1	NAS	NHL	L	78	11	6
carteje01	2006	1	PHI	NHL	C	62	14	2
ribeimi01	2009	1	DAL	NHL	C	66	19	3

Fig. 10: Example data identifying data inconsistency in Table ScoringShootout (marked red)

As a summary, we may visualize the impact of Entity/Relationship Profiling, just on the few tables we have considered in this use case. Fig. 11 and Fig. 12 show the access and linkage before and after E/R Profiling with our tool, respectively.

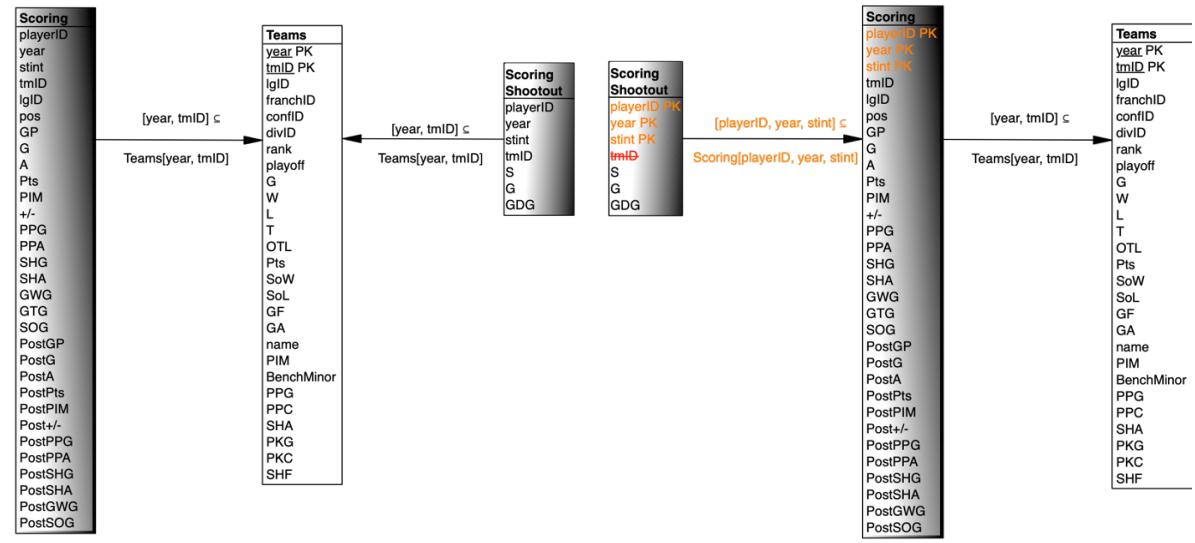


Fig. 11: Before E/R Profiling

Fig. 12: After E/R Profiling

Before Profiling, neither Scoring nor ScoringShootout were accessible since no primary key was specified. Hence, entity integrity could not be guaranteed. Entity Profiling brought forward the primary key *{playerID, year, stint}* for both tables, providing a natural mechanism to enforce entity integrity and efficient access to any records in that table via the UNIQUE index generated from the primary key. In addition, the foreign key from ScoringShootout to Scoring using the common key *{playerID, year, stint}* was identified. Together with the existing foreign key *Scoring[tmID, year]* $\subseteq$ *Teams[tmID, year]*, the new foreign key/key relationship made the old foreign key *ScoringShootout[tmID, year]* $\subseteq$ *Teams[tmID, year]* redundant, as well as the field *tmID* on *ScoringShootout*.

5. CLOSING

In closing, the *DataViadotto Profiler* helps your team discover all opportunities for accessing and linking data effectively, as a foundation for aligning data and enterprise models to maximize the acquisition of value from data.

ABOUT DATAVIADOTTO

DataViadotto is the industry pioneer for Entity/Relationship profiling technology. The company draws on decades of academic research to make the process of discovering models from data more effective, efficient and intuitive. Ultimately, data becomes profitable.

FOR ADDITIONAL QUESTIONS, CONTACT DATAVIADOTTO

www.viadotto.tech

