

Primary, Surrogate and Business Keys: the Semantics and Syntax of Codd's Information Principle



It all started with Aristotle.

But that's a story for another time....

**The DAMA International Symposium &
Wilshire Meta-Data Conference
San Diego, CA.
March, 2008.**

**Dr. Tom Johnston
Principal Consultant
Mindful Data, Inc.
tjohnston@mindfuldata.com**

Agenda

Note: in the announcement for this presentation, I said that some remarks by Mr. C. J. Date indicated that he believed that the use of surrogate keys violated the Information Principle. That was incorrect, and I apologize to Mr. Date and to my audience for this error.



Proposed:

- That all primary keys (PKs) should be surrogate keys (SKs).
- That no business keys (BKs) should be surrogate keys (SKs).

The Argument from Practice:
total cost of ownership.

The Argument from Theory:
Syntax, semantics and homonyms.

FOPL: first-order predicate logic.

The Standard Objections:

- to much space.
- too much time.
- too much trouble.
- can't get there from here.

Codd's Information Principle:

- Some deficiencies
 - Bags
 - Reasoning about types.
- Some extensions:
 - Database schemas in FOPL.
 - Data dictionaries in FOPL.
 - Beyond FOPL.

Discussion and Wrap-Up.



The Proposal: Some Definitions

- That all primary keys (PKs) should be surrogate keys (SKs).
- That no business keys (BKs) should be surrogate keys (SKs).

Primary key: the column or columns recognized by the DBMS as the unique identifier for a row in a database table, and also as what its foreign keys (FKs) refer to.

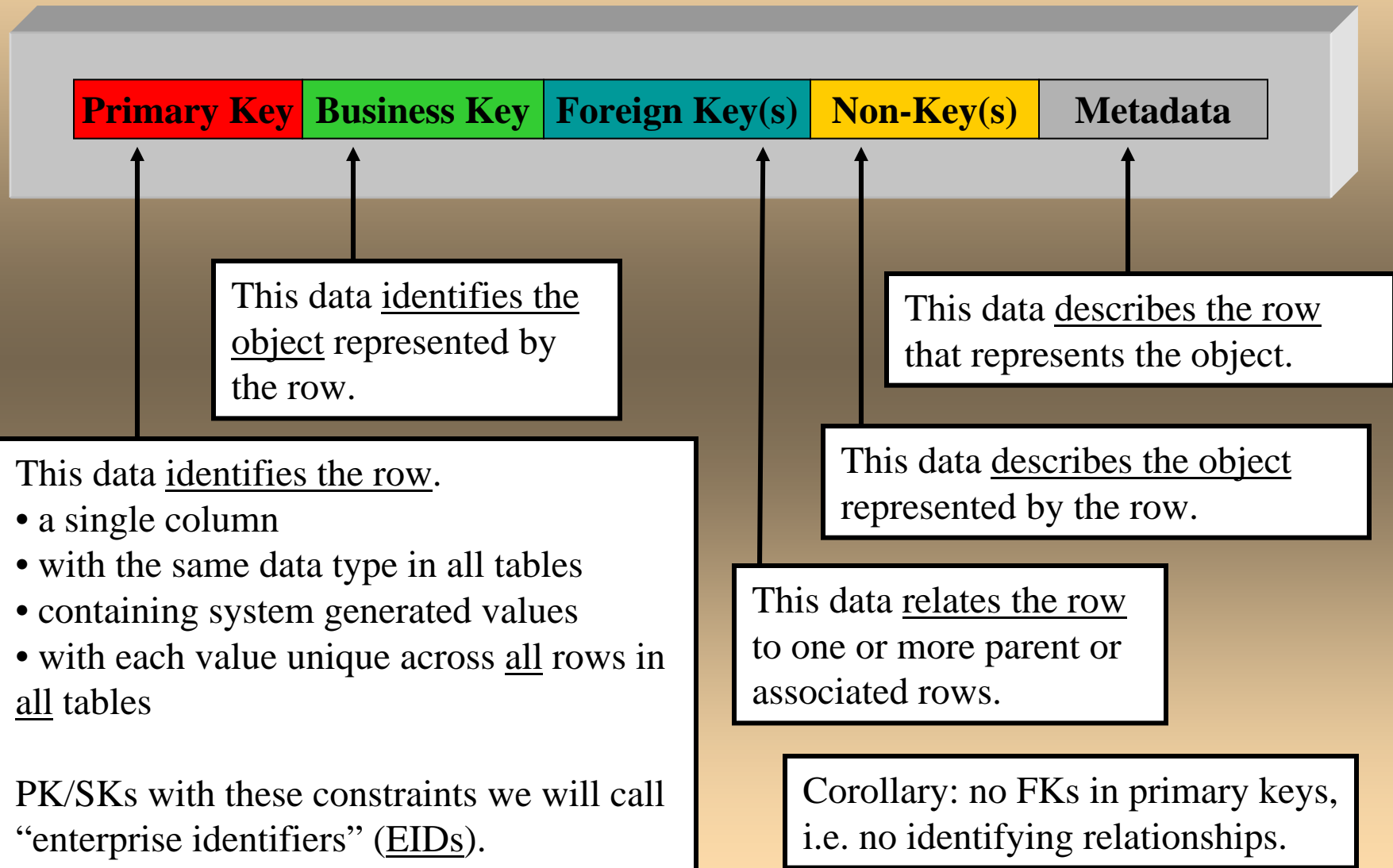
Surrogate key: a system-generated unique identifier for a row in a database table. It has no connection with the object represented by the row, i.e. no descriptive content.

Business key: a user-designated unique identifier for an object represented by a row in a database table. Either a name of the represented object, a description of it, or both.

Intelligent key: a single-column primary key in which two or more substrings are defined on separate domains.

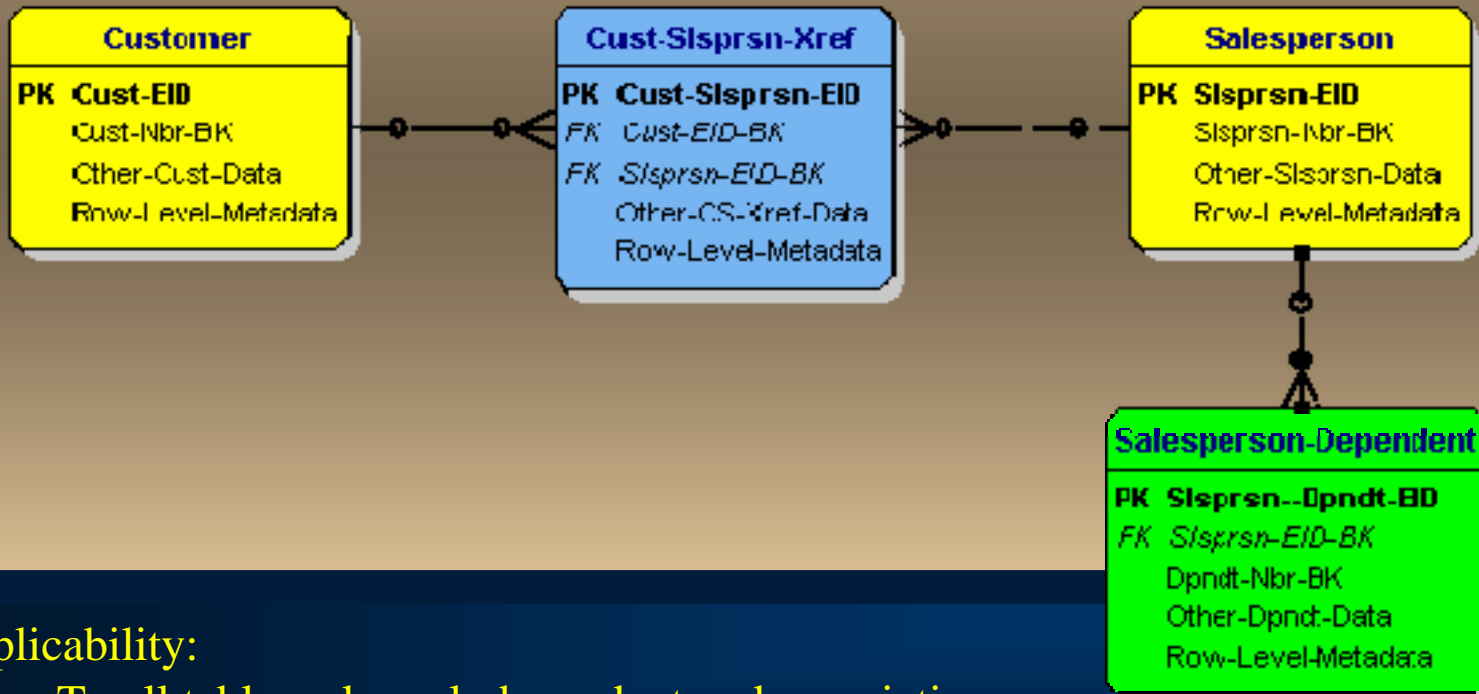


The Proposal: Five Types of Attributes





The Proposal: Scope



Applicability:

- To all tables – kernel, dependent and associative.
- Everywhere – to OLTP DBs, data warehouses, data marts.



The Argument from Practice: Case Study 1

Case Study 1:

When key value changes are non-atomic transactions, i.e. when there are too many to complete in a single batch run.

Change all queries to use a key-pair table.

Concurrently, begin replacing old values with new ones.

old value	new value
A18-CY	BB-421
.....
.....
193-S5	88Z-MH

When all values are converted, remove the key-pair table and change all queries back to their original state.

And when multiple databases, and federated queries are involved

Then “when all values are converted” means “when all instances of all values, across all databases are converted”.



The Argument from Practice: Case Study 2

Case Study 2:

How key type changes create even greater costs.

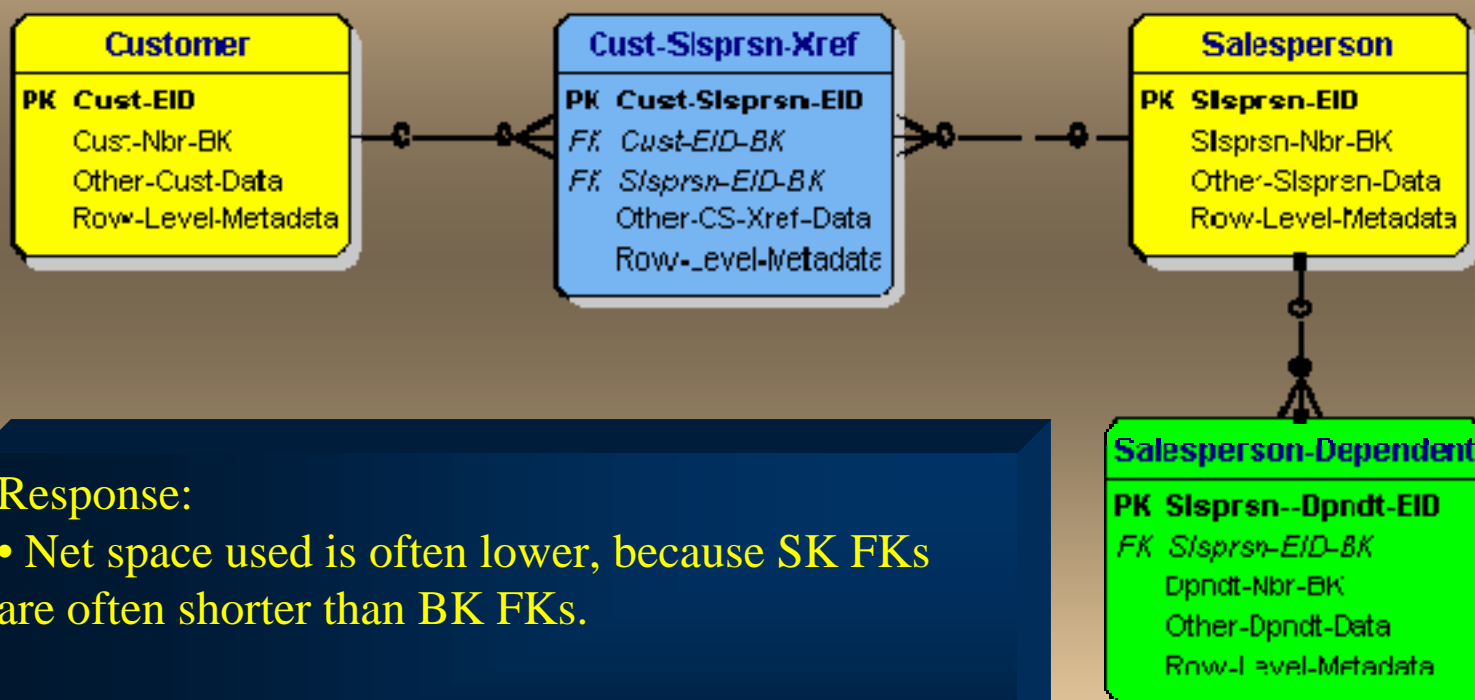
The business says: up till now, customer numbers have been xxx-yy-99999, with yy representing a region code. We need to expand the region code to three positions, and split the NY region into NY1 and NY2. How long will it take you to get that done?

- This business key is obviously an “intelligent key”. So another reason to avoid using BKs as PKs is that BKs are often intelligent. Intelligent keys are always at risk for business requests like the one above.
- If BKs are not PKs, BK data type changes affect only one table. But when BKs are PKs, and especially when this problem is made worse by the use of identifying relationships, BK data type changes can easily affect a dozen tables.
- And because these are not just value changes, a key pair file will not help us. These changes cannot be spread over multiple sessions during which the database is open for business. In situations like this, the cost differential between using the BK as a PK, and using an SK instead, can easily be orders of magnitude!



The Standard Objections : Too Much Space

Rows with SKs are longer than rows without them.



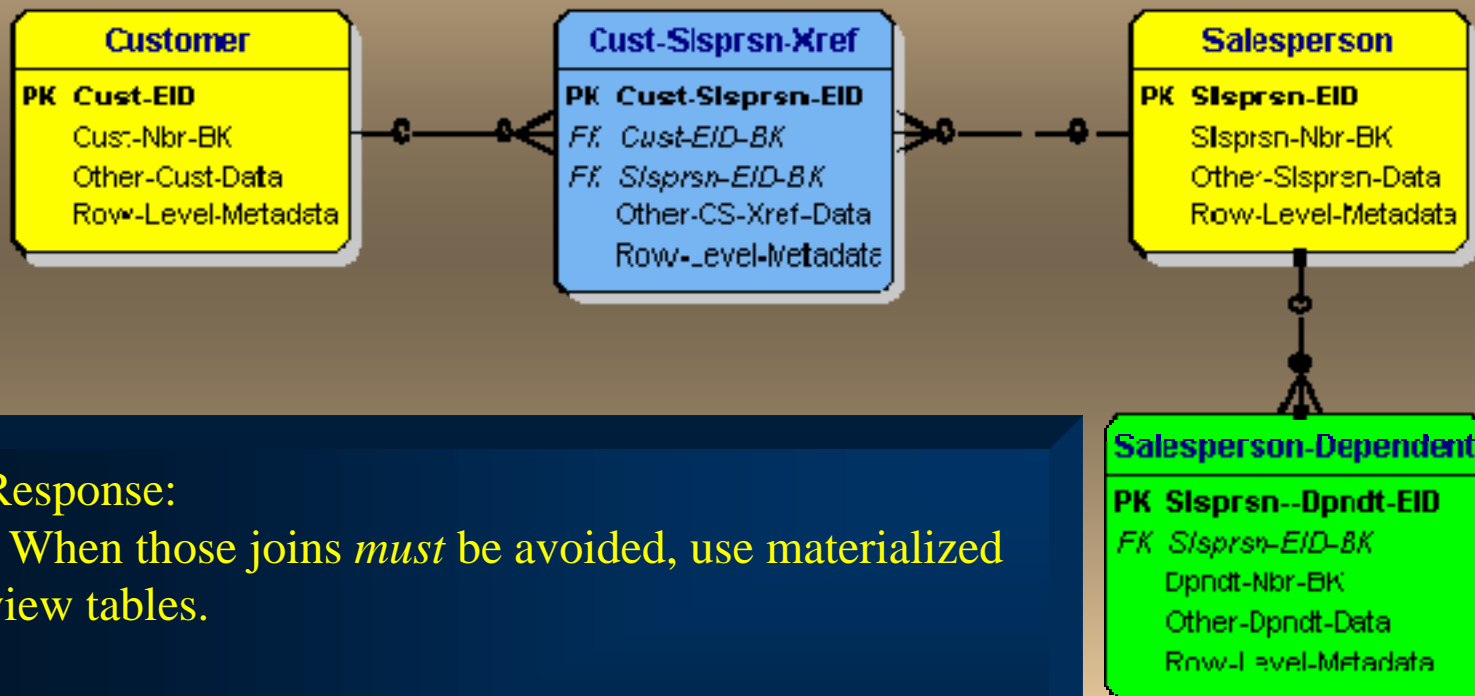
Response:

- Net space used is often lower, because SK FKs are often shorter than BK FKs.
- DASD space is an insignificant objection to a proposal that, if implemented, can drastically lower total cost of ownership.



The Standard Objections: Too Much Time

SK FKs mean extra joins to get BKs.



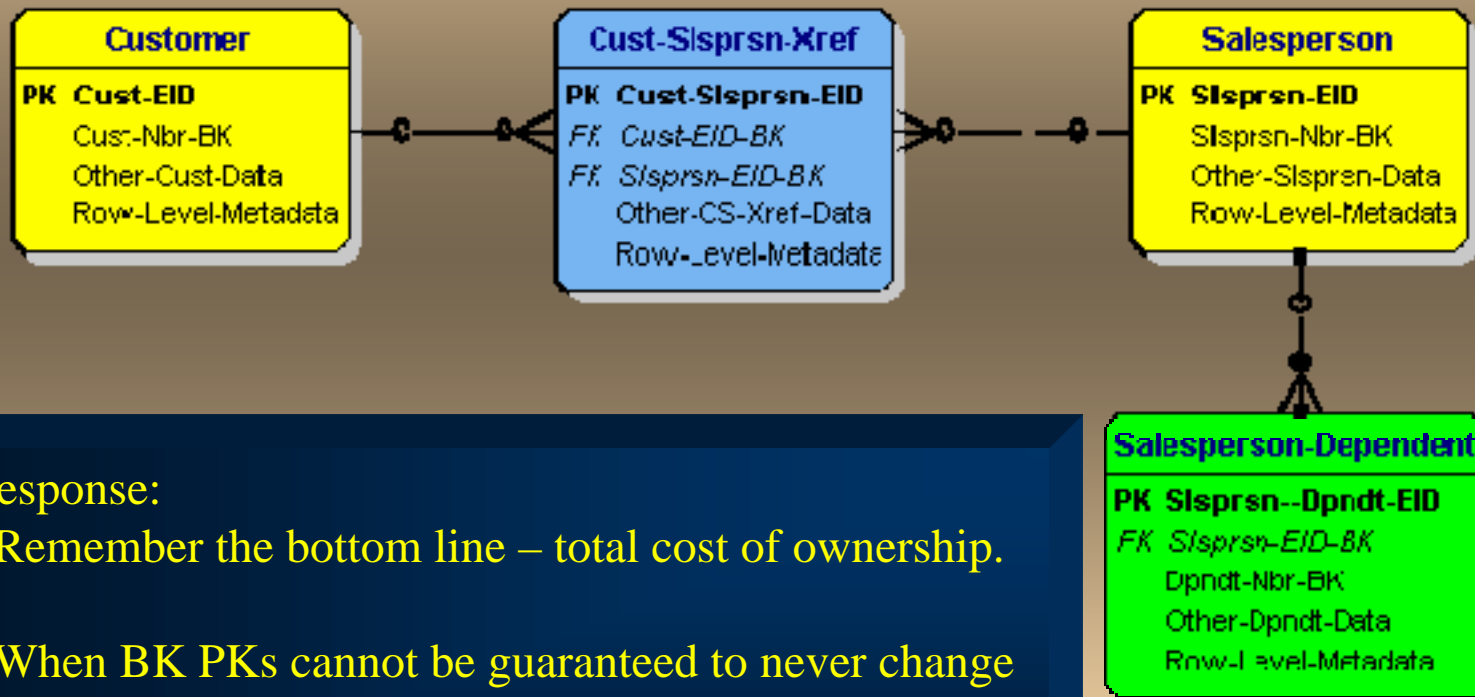
Response:

- When those joins *must* be avoided, use materialized view tables.
- At all costs, avoid propagating BKs to other *stored* tables, because that will defeat the entire purpose of keeping BKs and PKs separate, thus recreating the high total cost of ownership problem.



The Standard Objections: Too Much Trouble

With SKs, we're doing work the DBMS should be doing.



Response:

- Remember the bottom line – total cost of ownership.
- When BK PKs cannot be guaranteed to never change (and when can that be guaranteed?), then the cost of a single episode of change to BK PKs can easily exceed the original development cost.



The Standard Objections: Can't Get There From Here

A great idea, but we'll never be able to convert all our tables to EIDs.

- Response: the most formidable objection of all. But here's one approach, that can be implemented as gradually as you like.
 - Assign EIDs in the warehouse.
 - Put them as a match column in warehouse data sources, i.e. if those sources have BK PKs, just add the EID as a non-key match column for subsequent warehouse updating.
 - Propagate that match column everywhere there is a FK to its table.
 - Now BK PKs and their FKs are paired, row for row and value for value, by EIDs and their appearance as match columns on all affected tables.
 - Make the switch! Now it's just a matter of changing DDL.



The Argument from Theory: BK/PKs are Functional Homonyms

Semantic Functions.

- The function of primary keys: to identify.
- The function of foreign keys: to relate.
- The function of non-keys: to describe.

- So BK PKs both identify and describe.
- And BK FKs both relate and describe.
- Because they describe, the business may require changes to individual values, or to entire data types, so that they can describe more accurately.
- If BKs are separate from PKs, those changes affect only a single column of a single table.



Deficiencies of the Information Principle: The Semantics and Syntax of Keys.

Semantics, Syntax and the Information Principle.

- SKs and relations.
- BKs and bags.
- Is the Information Principle violated?

Is a table with an EID and a *non-unique* business key

- a relation, or
- a bag (aka a multi-set, aka a table with duplicates)?

Syntactically (to the DBMS), it's a relation.

Semantically (to the user), it's a table with duplicates, i.e. a bag.

Conclusion: the relational model does not support the semantics of bags.

To do so, we are forced to separate semantics from syntax, and manage the semantics ourselves.

But to the extent that the implementation of relational theory is relegated to the management of syntax only, Codd's Information Principle is an empty claim.



Deficiencies of the Information Principle: Managing the Semantics of Types.

- The relational model, and its implementations, is focused on managing *instances* of types, e.g. rows in tables, not the tables themselves.
- DDL does defines the syntax of tables. But it does not define their semantics.

- For instance, the relational model does not define inheritance across supertype/subtype hierarchies.
- Mathematically, this is the failure to manage properties of relationships – the transitive property in this case. Symmetry and reflexivity are other properties, and specific relationships will have specific additional semantic properties.

Conclusion: the relational model does not support the semantics of relationships.

As we all know, the DBMS doesn't care about relationship names. Yet these names carry important information.

To manage the semantics of relationships, we are forced to manage them ourselves. And so, once again, we write code!

But to the extent that the implementation of relational theory is relegated to the management of syntax only, Codd's Information Principle is an empty claim.



Codd's Information Principle, Extended

The Information Principle Extended.

New structures + new transformations → more powerful inferencing (formalized reasoning).

Are relational structures and relational transformations able to express them all?

What are these new structures?

What is this new inferencing?

The relational model specifies inferencing (queries) about instances. Relational queries use a subset of FOPL, called “existential conjunctive” logic.

The next generation of inferencing engines are being developed to support inferencing about types, and will require full FOPL.



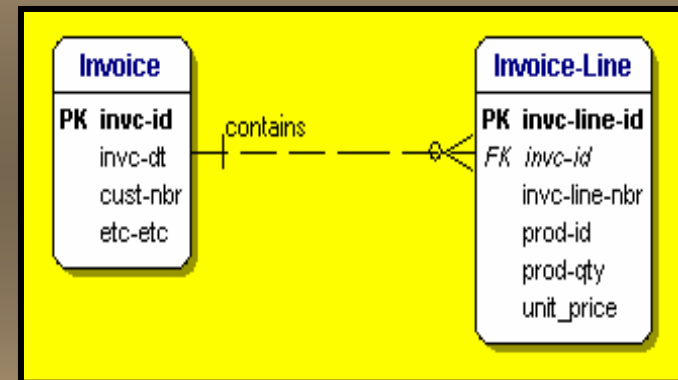
An Extension: Expressing Database Schemas in FOPL

Example: a minimum cardinality constraint.

In English: “Every invoice line must be contained by one and only one invoice”.

In SQL DDL: expressed as a non-nullable foreign key from invoice line to invoice.

In English/logic: “For all x, if x is an invoice line, then there exists a y such that y is an invoice and y contains x; and for all z, if z is an invoice and z contains x, then z=y”.



In FOPL, with IL = “Invoice Line”; I = “Invoice”; and C = “Contains”:

$$\forall x[IL(x) \rightarrow \exists y[I(y) \ \& \ C(y,x) \ \& \ [\forall z[I(z) \ \& \ C(z,x) \rightarrow (z=y)]]]]$$



An Extension: Data Dictionaries and Controlled Vocabularies

Data Dictionary: a good definition. Customer: a [person] or [organization] who, within the last five years, has either responded to {marketing material} or made an {invoiced purchase} from our company.

Data Dictionary.
Person

Data Dictionary.
Organization

Controlled Vocabulary.
Marketing Material
.....

Controlled Vocabulary.
Invoiced Purchase



Another Extension: Expressing Data Definitions in FOPL

In English: (def) Customer: a [person] or [organization] who, within the last five years, has either responded to {marketing material} or made an {invoiced purchase} from our company.

In English/logic: for all c , if c is a customer, then either c is a person or c is an organization, and there exists an event e such that either e is a marketing response or e is an invoiced purchase, and c took part in e , and there is a time t such that t is between now and five years ago, and e occurred at t .

$$\forall c(\text{Cust}(c)) \rightarrow \{ \text{Per}(c) \text{ or } \text{Org}(c) \text{ and } \exists e(\text{MktRspnse}(e) \text{ or } \text{InvPrchs}(e)) \text{ and } \text{TookPartIn}(c,e) \text{ and } \exists t(\text{GT}(t,(\text{now}-5\text{yrs}) \text{ and } \text{LT}(t,\text{now})) \text{ and } \text{Occrd-At}(e,t)) \}$$



How the Relational Model Fails Codd's Information Principle: Background (1)

In the relational model, the semantics of description are layered onto the syntax of data types, domains, and the interpretations given to domain values. This implementation of a specific kind of semantics in a specific kind of syntax has led to no harm, because descriptive semantics are independent of mathematical transformations. They just describe something already picked out.

Business keys do (and should) uniquely pick out the things represented by the rows of their tables. But it is common to use business keys as primary keys, in which case they are also used to uniquely pick out those very rows themselves.

And that's the problem.



How the Relational Model Fails Codd's Information Principle: Background (2)

The use of business keys as primary keys is what allows RDBMSs – engines that enforce the constraints specified by the relational model – to enforce the semantics of entity integrity and referential integrity. Without this conflation of business keys with primary keys, RDBMSs cannot enforce these two critical constraints.

But, as I have argued in this presentation, this conflates the roles of identifying and describing, and these roles then often conflict. When they do, and changes are required because of changes in the descriptive semantics of those keys, the cost of change is usually very high, often exceeding the cost of initial development and implementation.

So we must eliminate the homonyms. We must create *surrogate* keys to uniquely identify individual rows of tables, and to join rows together.

But then the semantics of entity and referential integrity cannot be enforced by the DBMS. This is a heavy semantic burden, and it must be enforced outside the relational model and its implementations, by us, with code.



How the Relational Model Fails Codd's Information Principle: Current Shortcomings

Bags. The relational model explicitly disavows multi-sets. But businesses need them. Data in the real world, including keys, is often not as pristine as theorists imagine.

Relationship types. The relational model does not support the semantics of relationship types. In particular, it does not support the semantics of inheritance from supertypes by subtypes. We developers must do it, with code.

Types in general. The relational model does not support the semantics of any types. Reflected in the subset of FOPL that it warrants, the relational model focuses on managing *instances* of types. It is not able to recognize and use the properties of reflexivity, symmetry and transitivity as they apply to predicates which represent relationships. We developers must support these semantics, with code.

Semantic entity integrity. The relational model supports it only by creating functional homonyms. The cost, when seen clearly, is unacceptable. We developers must provide this support, with code.

Semantic referential integrity. Ipso facto, the relational model does not support semantic referential integrity. We developers must do it, with code.



How the Relational Model *Will* Fail Codd's Information Principle: Future Developments (1)

And what of the future? The relational model does not begin to address more advanced formalizations of inferencing, such as:

- Full FOPL. The formalization of algorithmically expressed semantics in full FOPL (leading to the formalization of business rules, and of business definitions).
- Higher-order predicate logics. Quantification over predicates. This is another step in the formalization of reasoning about types.
- Multi-valued logic. The proper formalization of operations involving {null} or {null}s. (This is not the formalization we currently have.)



How the Relational Model *Will* Fail Codd's Information Principle: Future Developments (2)

- Modal logic. The formalization of reasoning involving distinctions between what is necessary and what is possible. Also, the formalization of reasoning involving past and future (and compound tenses as well).
- Fuzzy logic. The quantification of vagueness.
- Inductive logic. Reasoning with probabilities and inverse probabilities.
- Choice theory. The logic of rational decision-making. The expansion and contraction conditions. Paul Samuelson's revelation condition.
- Analogical reasoning. As in Islamic law. (This is recent work being done by Dr. John Sowa, which he presented at last year's Semantic Technology Conference.)



Codd's Information Principle: a Final Word

The Information Principle: the *Letter* of the Law.

All information should be expressed in a specific syntax and grammar – the syntax of relations (named subsets of the Cartesian Product of an unordered set of named ordered or unordered sets), and the grammar defining well-formed sentences (SQL statements) and the relational algebra – a subset of FOPL – defining valid transformations of those grammatical elements.

In this sense, Codd's Information Principle is already obsolete, and will only become more so as advances in formalization continue.

The Information Principle: the *Spirit* of the Law.

All semantics should be expressed in explicit structures and transformations that are mathematically rigorous, and that form a closed system. The remaining semantics should reside in the interpretation of the formalism, provided by its users, and nowhere else.

In this sense, Codd's Information Principle continues to guide us.



Wrap-up and Discussion

One proposal has been that all primary keys should be surrogate keys, and indeed a special kind of surrogate key, an EID.

Would anyone like to describe a situation in which they believe that surrogate keys should not be used?

A second proposal has been that Codd's Information Principle is incorrect if interpreted as the statement that all knowledge can be represented by relations, and persisted and maintained in the structures and transformations defined by the relational model.

The suggestion has been that Codd's Information Principle can be interpreted as saying that inference engines of any kind (and not just relational DBMSs) can reliably produce information from data only if the data structures and transformations are mathematically rigorous, and form a closed system.

Would anyone like to make the case that this suggestion is being too liberal with the Information Principle, and that Codd's principle will remain true in the strict sense, no matter how advanced our DBMSs and other inference engines become?