CAS CS 460/660
Introduction to Database Systems

Functional Dependencies
and
Normal Forms

Review: Database Design

- Requirements Analysis
  - user needs; what must database do?
- Conceptual Design
  - high level descr (often done w/ER model)
- Logical Design
  - translate ER into DBMS data model
- Schema Refinement
  - **consistency,normalization**
- Physical Design - indexes, disk layout
- Security Design - who accesses what

Keys (review)

- A key is a set of attributes that uniquely identifies each tuple in a relation.
- A candidate key is a key that is minimal.

  If AB is a candidate key, then neither A nor B is a key on its own.

- A superkey is a key that is not necessarily minimal (although it could be)

  If AB is a candidate key then ABC, ABD, and even AB are superkeys.

(Review) Projection

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

$$\pi_{sname,rating}(S2)$$

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

S2

age
35.0
55.5

$$\pi_{age}(S2)$$

Functional Dependencies (FDs)

- A <u>functional dependency</u> $X \rightarrow Y$ holds over relation schema R if, for every allowable instance *r* of R:

$$t1 \in r,\ \ t2 \in r,\ \ \pi_X(t1) = \pi_X(t2)$$
$$\text{implies} \quad \pi_Y(t1) = \pi_Y(t2)$$

(where t1 and t2 are tuples;X and Y are sets of attributes)

- In other words: $X \rightarrow Y$ means

  Given any two tuples in *r*, if the X values are the same, then the Y values must also be the same. (but not vice versa)

- Can read "$\rightarrow$" as "determines"

FD's Continued

- An FD is a statement about *all* allowable relations.
  - Identified based on application semantics
  - Given some instance *r1* of R, we can check if *r1* violates some FD *f*, but we cannot determine if *f* holds over R.
- How related to keys?
  - if "K → all attributes of R" then

    K is a *superkey* for R

    (does not require K to be *minimal*.)
  - FDs are a generalization of keys.

Example: Constraints on Entity Set

- Consider relation obtained from Hourly_Emps:

Hourly_Emps (*<u>ssn</u>, name, lot, rating, wage_per_hr, hrs_per_wk*)

  - We sometimes denote a relation schema by listing the attributes: e.g., SNLRWH
  - This is really the *set* of attributes {S,N,L,R,W,H}.
  - Sometimes, we refer to the set of *all attributes* of a relation by using the relation name. e.g., "Hourly_Emps" for SNLRWH

- What are some FDs on Hourly_Emps (Given)?

ssn is the key: S → SNLRWH

rating determines *wage_per_hr*: R → W

lot determines *lot*: L → L ("trivial" dependnency)

Redundancy Problems Due to R → W

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

Hourly_Emps

- *Update anomaly*: Can we modify W in only the 1st tuple of SNLRWH?
- *Insertion anomaly*: What if we want to insert an employee and don't know the hourly wage for his or her rating? (or we get it wrong?)
- *Deletion anomaly*: If we delete all employees with rating 5, we lose the information about the wage for rating 5!

Detecting Reduncancy

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

Hourly_Emps

Q: Why is R → W problematic, but S→W not?

Taming Schema Redundancy

- Integrity constraints, in particular *functional dependencies*, can be used to identify schemas with such problems and to suggest refinements.

- Main refinement technique: *decomposition*
  - replacing ABCD with, say, AB and BCD, or ACD and ABD.

- Decomposition should be used judiciously:
  - Is there reason to decompose a relation?
  - What problems (if any) does the decomposition cause?

Decomposing a Relation

- Redundancy can be removed by "chopping" the relation into pieces.
- FD's are used to drive this process.

  R → W is causing the problems, so decompose SNLRWH into what relations?

S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

Hourly_Emps2

R	W
8	10
5	7

Wages

Reasoning About FDs

- Given some FDs, we can usually infer additional FDs:

  title → *studio, star* **implies** *title* → *studio* and *title* → *star*

  title → *studio* and *title* → *star* **implies** *title* → *studio, star*

  title → *studio*, *studio* → *star* **implies** *title* → *star*

  But,

  title, star → *studio* **does NOT necessarily imply that** *title* → *studio* or that *star* → *studio*

- An FD *f* is *implied by* a set of FDs *F* if *f* holds whenever all FDs in *F* hold.

- F^+ = *closure of F* is the set of all FDs that are implied by *F*. (includes "trivial dependencies")

Rules of Inference

- Armstrong's Axioms (X, Y, Z are sets of attributes):
  - *Reflexivity*: If $Y \subseteq X$, then $X \rightarrow Y$
  - *Augmentation*: If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
  - *Transitivity*: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$
- These are *sound* and *complete* inference rules for FDs!
  - i.e., using AA you can compute all the FDs in F+ and only these FDs.
- Some additional rules (that follow from AA):
  - *Union*: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
  - *Decomposition*: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

Example

- Contracts(<u>*cid*</u>,*sid,jid,did,pid,qty,value*), and:
  - C is the key: C → CSJDPQV
  - Job purchases each part using single contract: JP → C
  - Dept purchases at most 1 part from a supplier: SD → P
- Problem: Prove that SDJ is a key for Contracts
  - JP → C, C → CSJDPQV imply JP → CSJDPQV

    (by transitivity) (shows that JP is a key)
  - SD → P implies SDJ → JP (by augmentation)
  - SDJ → JP, JP → CSJDPQV imply SDJ → CSJDPQV
  - (by transitivity) thus SDJ is a key.

Q: can you now infer that SD → CSDPQV (i.e., drop J on both sides)?

No! FD inference is not like arithmetic multiplication.

Attribute Closure

- Size of F^+ is exponential in # attributes in R;
  - Computing it can be expensive.
- If we just want to check if a given FD $X \rightarrow Y$ is in F^+, then:

  1) Compute the *attribute closure* of X (denoted X^+) wrt F
     - X^+ = Set of all attributes A such that $X \rightarrow A$ is in F^+
       - initialize $X^+ := X$
       - Repeat until no change:

         if $U \rightarrow V$ in F such that U is in X^+, then add V to X^+

  2) Check if Y is in X^+

- Can also be used to find the keys of a relation.
  - If all attributes of R are in X^+ then X is a superkey for R.
  - Q: How to check if X is a "candidate key"?

Attribute Closure (example)

- $R = \{A, B, C, D, E\}$
- $F = \{ B \rightarrow CD, D \rightarrow E, B \rightarrow A, E \rightarrow C, AD \rightarrow B \}$
- Is $B \rightarrow E$ in F^+ ?

  $B^+ = B$

  $B^+ = BCD$

  $B^+ = BCDA$

  $B^+ = BCDAE$ ... Yes! B is a key for R too!
- Is D a key for R?

  $D^+ = D$

  $D^+ = DE$

  $D^+ = DEC$

  ... Nope!

- **Is AD a key for R?**

  $AD^+ = AD$

  $AD^+ = ABD$ and B is a key, so Yes!
- **Is AD a *candidate* key for R?**

  $A^+ = A$

  A not a key, nor is D so Yes!
- **Is ADE a *candidate* key for R?**

  No! AD is a key, so ADE is a superkey, but not a cand. key

Normal Forms

- Question: is any refinement needed??!
- If a relation is in a *normal form* (BCNF, 3NF etc.):
  - we know that certain problems are avoided/minimized.
  - helps decide whether decomposing a relation is useful.
  - NFs are syntactic rules (don't need to understand app)
- Role of FDs in detecting redundancy:
  - Consider a relation R with 3 attributes, ABC.
    - No (non-trivial) FDs hold: There is no redundancy here.
    - Given $A \rightarrow B$: If A is not a key, then several tuples could have the same A value, and if so, they' ll all have the same B value!
- 1st Normal Form – all attributes are atomic (i.e., "flat tables")
- 1st $\supset$ 2nd (of historical interest) $\supset$ 3rd $\supset$ Boyce-Codd $\supset$ …

Normal Forms

Normal form	Defined by	Brief definition
First normal form (1NF)	Two versions: E.F. Codd (1970), C.J. Date (2003)[9]	Table faithfully represents a relation and has no *repeating groups*
Second normal form (2NF)	E.F. Codd (1971)[2]	No non-prime attribute in the table is functionally dependent on a proper subset of any candidate key
Third normal form (3NF)	E.F. Codd (1971);[2] see also Carlo Zaniolo's equivalent but differently expressed definition (1982)[10]	Every non-prime attribute is non-transitively dependent on every candidate key in the table. The attributes that do not contribute to the description of the primary key are removed from the table. In other words, no transitivity dependency is allowed.
Elementary Key Normal Form (EKNF)	C.Zaniolo (1982)[10]	Every non-trivial functional dependency in the table is either the dependency of an elementary key attribute or a dependency on a superkey
Boyce–Codd normal form (BCNF)	Raymond F. Boyce and E.F. Codd (1974)[11]	Every non-trivial functional dependency in the table is a dependency on a superkey
Fourth normal form (4NF)	Ronald Fagin (1977)[12]	Every non-trivial multivalued dependency in the table is a dependency on a superkey
Fifth normal form (5NF)	Ronald Fagin (1979)[13]	Every non-trivial join dependency in the table is implied by the superkeys of the table
Domain/key normal form (DKNF)	Ronald Fagin (1981)[14]	Every constraint on the table is a logical consequence of the table's domain constraints and key constraints
Sixth normal form (6NF)	C.J. Date, Hugh Darwen, and Nikos Lorentzos (2002)[15]	Table features no non-trivial join dependencies at all (with reference to generalized join operator)

Boyce-Codd Normal Form (BCNF)

- Reln R with FDs *F* is in BCNF if, for all $X \rightarrow A$ in F^+
  - $A \in X$ (called a *trivial* FD), or
  - X is a superkey for R.
- In other words: "R is in BCNF if the only non-trivial FDs over R are *key constraints*."
- If R in BCNF, then every field of every tuple records information that cannot be inferred using FDs alone.
  - Say we are told that FD $X \rightarrow A$ holds for this example relation:

• Can you guess the value of the missing attribute?

•Yes, so relation is not in BCNF

X	Y	A
x	y1	a
x	y2	?

Boyce-Codd Normal Form - Alternative Formulation

"The key, the whole key, and nothing but the key"

Decomposition of a Relation Scheme

- If a relation is not in a desired normal form, it can be *decomposed* into multiple relations that each are in that normal form.

- Suppose that relation R contains attributes *A1 ... An.* A decomposition of R consists of replacing R by two or more relations such that:
  - Each new relation scheme contains a subset of the attributes of R, and
  - Every attribute of R appears as an attribute of at least one of the new relations.

Example

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

Hourly_Emps

- SNLRWH has FDs $S \rightarrow SNLRWH$ and $R \rightarrow W$
- Q: Is this relation in BCNF?

No, The second FD causes a violation;
W values repeatedly associated with R values.

Decomposing a Relation

- Easiest fix is to create a relation RW to store these associations, and to remove W from the main schema:

S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

Hourly_Emps2

R	W
8	10
5	7

Wages

•Q: Are both of these relations now in BCNF?

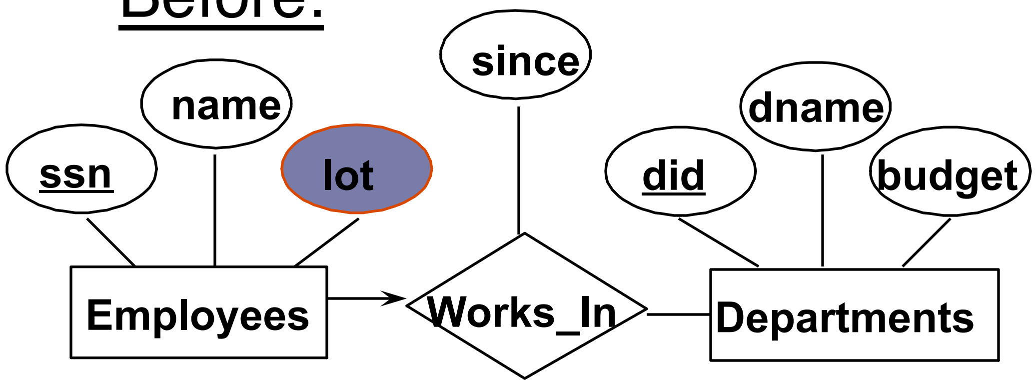
•**Decompositions should be used only when needed.**

–Q: potential problems of decomposition?

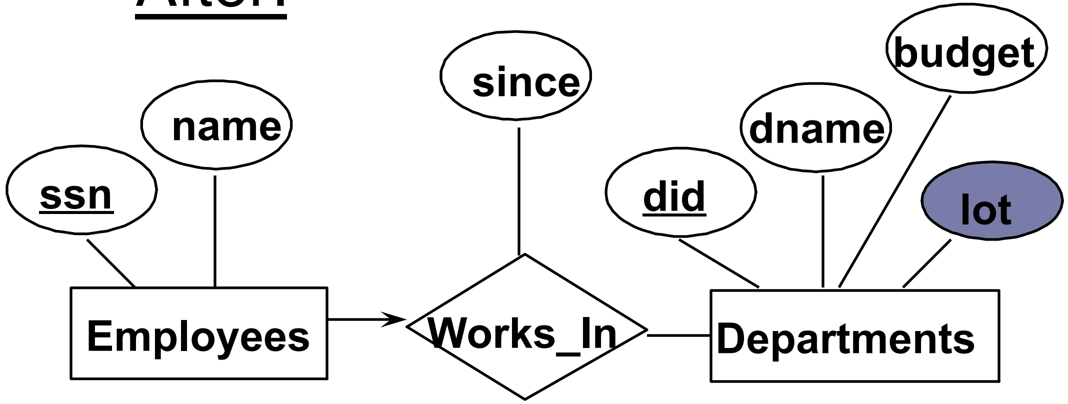
Refining an ER Diagram

- 1st diagram becomes:
  Workers(S,N,L,D,Si)
  Departments(D,M,B)
  - Lots associated with workers.
- Suppose all workers in a dept are assigned the same lot: D → L
- Redundancy; fixed by:
  Workers2(S,N,D,Si)
  Dept_Lots(D,L)
  Departments(D,M,B)
- Can fine-tune this:
  Workers2(S,N,D,Si)
  Departments(D,M,B,L)

Before:



After:



Decomposing a Relation

- Easiest fix is to create a relation RW to store these associations, and to remove W from the main schema:

S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

Hourly_Emps2

R	W
8	10
5	7

Wages

•Q: Are both of these relations now in BCNF?

•**Decompositions should be used only when needed.**

–Q: potential problems of decomposition?

Problems with Decompositions

- There are three potential problems to consider:
  1) May be impossible to reconstruct the original relation! (Lossiness)
     - Fortunately, not in the SNLRWH example.
  2) Dependency checking may require joins.
     - Fortunately, not in the SNLRWH example.
  3) Some queries become more expensive.
     - e.g., How much does Guldu earn?

Lossiness (#1) cannot be allowed

#2 and #3 are design tradeoffs: Must consider these issues vs. redundancy.

(Review) Rel Alg Operator: Join ($\bowtie$)

- Joins are compound operators involving cross product, selection, and (sometimes) projection.

- Most common type of join is a "*natural join*" (often just called "join"). R $\bowtie$ S conceptually is:
  - Compute R X S
  - Select rows where attributes that appear in both relations have equal values
  - Project all unique attributes and one copy of each of the common ones.

- Note: Usually done much more efficiently than this.
- Useful for putting "normalized" relations back together.

Natural Join Example

sid	bid	day
22	101	10/10/96
58	103	11/12/96

R1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S1

R1 $\bowtie$ S1 =

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96

Lossless Decomposition (example)

S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

$\bowtie$

R	W
8	10
5	7

=

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

Lossy Decomposition (example)

A	B	C
1	2	3
4	5	6
7	2	8

$\Rightarrow$

A	B
1	2
4	5
7	2

B	C
2	3
5	6
2	8

$A \rightarrow B; C \rightarrow B$

A	B
1	2
4	5
7	2

$\bowtie$

B	C
2	3
5	6
2	8

$=$

A	B	C
1	2	3
4	5	6
7	2	8
1	2	8
7	2	3

Lossless Decomposition

- Decomposition of R into X and Y is *lossless-join* w.r.t. a set of FDs F if, for every instance *r* that satisfies F:

  $$\pi_X(r) \bowtie \pi_Y(r) = r$$

- The decomposition of R into X and Y is lossless with respect to F *if and only if* F^+ contains:

  $$X \cap Y \rightarrow X, \quad \textbf{or}$$

  $$X \cap Y \rightarrow Y$$

  in previous example: decomposing ABC into AB and BC is lossy, because intersection (i.e., "B") is not a key of either resulting relation.

- Useful result: If $W \rightarrow Z$ holds over R and $W \cap Z$ is empty, then decomposition of R into R-Z and WZ is lossless.

Lossless Decomposition (example)

A	B	C
1	2	3
4	5	6
7	2	8

$\Rightarrow$

A	C
1	3
4	6
7	8

B	C
2	3
5	6
2	8

$A \rightarrow B;\ C \rightarrow B$

A	C
1	3
4	6
7	8

$\bowtie$

B	C
2	3
5	6
2	8

$=$

A	B	C
1	2	3
4	5	6
7	2	8

But, now we can't check $A \rightarrow B$ without doing a join!

Dependency Preserving Decomposition

- Dependency preserving decomposition (Intuitive):
  - If R is decomposed into X, Y and Z, and we enforce the FDs that hold individually on X, on Y and on Z, then all FDs that were given to hold on R must also hold. *(Avoids Problem #2 on our list.)*

- The projection of F on attribute set X (denoted F_X) is the set of FDs

$U \rightarrow V$ in F^+ (*closure of F , not just F*) such that all of the attributes on both sides of the f.d. are in X.

  - *That is: U and V are subsets of X*

Dependency Preserving Decompositions (Contd.)

- Decomposition of R into X and Y is *dependency preserving* if

$$(F_X \cup F_Y)^+ = F^+$$

  - i.e., if we consider only dependencies in the closure F^+ that can be checked in X without considering Y, and in Y without considering X, these imply all dependencies in F^+.
- Important to consider F^+ in this definition:
  - ABC, $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow A$, decomposed into AB and BC.
  - Is this dependency preserving? Is $C \rightarrow A$ preserved?????
    - note: F^+ contains $F \cup \{A \rightarrow C, B \rightarrow A, C \rightarrow B\}$, so…
- F_{AB} contains $A \rightarrow B$ and $B \rightarrow A$; F_{BC} contains $B \rightarrow C$ and $C \rightarrow B$
- So, $(F_{AB} \cup F_{BC})^+$ contains $C \rightarrow A$

Decomposition into BCNF

- Consider relation R with FDs F.

  If $X \rightarrow Y$ violates BCNF, decompose R into R - Y and XY (guaranteed to be lossless).

  - Repeated application of this idea will give us a collection of relations that are in BCNF; lossless join decomposition, and guaranteed to terminate.
  - e.g., CSJDPQV, key C, $JP \rightarrow C$, $SD \rightarrow P$, $J \rightarrow S$
  - *{contractid, supplierid, projectid,deptid,partid, qty, value}*
  - To deal with $SD \rightarrow P$, decompose into SDP, CSJDQV.
  - To deal with $J \rightarrow S$, decompose CSJDQV into JS and CJDQV
  - So we end up with: SDP, JS, and CJDQV

- Note: several dependencies may cause violation of BCNF. The order in which we fix them could lead to very different sets of relations!

BCNF and Dependency Preservation

- In general, there may not be a dependency preserving decomposition into BCNF.
  - e.g., CSZ, CS → Z, Z → C
  - Can't decompose while preserving 1st FD; not in BCNF.
- Similarly, decomposition of CSJDPQV into SDP, JS and CJDQV is not dependency preserving (w.r.t. the FDs JP → C, SD → P and J → S).
- *{contractid, supplierid, projectid,deptid,partid, qty, value}*
  - However, it is a lossless join decomposition.
  - In this case, adding JPC to the collection of relations gives us a dependency preserving decomposition.
    - but JPC tuples are stored only for checking the f.d. (*Redundancy!*)

Third Normal Form (3NF)

- Reln R with FDs *F* is in 3NF if, for all $X \rightarrow A$ in F^+
  - $A \in X$ (called a *trivial* FD), or
  - X is a superkey of R, or
  - A is part of some candidate key (not superkey!) for R. (sometimes stated as "A is *prime*")
- *Minimality* of a key is crucial in third condition above!
- If R is in BCNF, obviously in 3NF.
- If R is in 3NF, some redundancy is possible. It is a compromise, used when BCNF not achievable (e.g., no ``good'' decomp, or performance considerations).
  - *Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations always possible.*

Decomposition into 3NF

- Obviously, the algorithm for lossless join decomp into BCNF can be used to obtain a lossless join decomp into 3NF (typically, can stop earlier) but does not ensure dependency preservation.
- To ensure dependency preservation, one idea:
  - If $X \rightarrow Y$ is not preserved, add relation XY.

  Problem is that XY may violate 3NF! e.g., consider the addition of CJP to `preserve' $JP \rightarrow C$. What if we also have $J \rightarrow C$?
- Refinement: Instead of the given set of FDs F, use a *minimal cover for F*.

Minimal Cover for a Set of FDs

- *Minimal cover* G for a set of FDs F:
  - Closure of F = closure of G.
  - Right hand side of each FD in G is a single attribute.
  - If we modify G by deleting an FD or by deleting attributes from an FD in G, the closure changes.
- Intuitively, every FD in G is needed, and ``*as small as possible*'' in order to get the same closure as F.
- e.g., $A \rightarrow B$, $ABCD \rightarrow E$, $EF \rightarrow GH$, $ACDF \rightarrow EG$ has the following minimal cover:
  - $A \rightarrow B$, $ACD \rightarrow E$, $EF \rightarrow G$ and $EF \rightarrow H$
- M.C. implies 3NF, Lossless-Join, Dep. Pres. Decomp!!!
  - (more in book)

Assertions

- How to test if and FD is satisfied?

- ASSERTIONS:

CREATE ASSERTION assertion_name CHECK predicate

Example:

```
CREATE ASSERTION SmallClub
CHECK ((SELECT COUNT(S.sid) FROM Sailors S) +
            (SELECT COUNT(B.bid) FROM Boats B) < 100)
```

Assertions

Constraint: A customer with a loan should have an account with at least 1000 dollars.

```
create assertion  balance_constraint check
 (not exists (select * from loan L
                  where not exists (select *
                              from borrower B, depositor D, account A
                        where L.loan_no =  B.loan_no
                        and B.cname = D.cname
                        and D.account_no = A.account_no
                              and A.balance >= 1000 ))
```

Another example

customer(customer_name, customer_street, customer_city)

Constraint: Customer city is always not null.

Can enforce it with an assertion:

```
Create Assertion CityCheck Check
    (   NOT EXISTS (
         Select   *
         From     customer
         Where  customer_city is null));
```