Consistency Constraints (enforcing correct data)

Objectives:

• The DBMS as an active manager of the correctness contents and behavior, enabling more object like behavior.

• Specification of such in SQL

["beer slides" borrowed from Garcia-Molina via Tim Finin]

Data Model

"A *data model* is a notation for describing data or information… generally consisting of three parts"

- Structure of the data…
- Operations on the data …
- Constraints on the data …

Context: In the beginning

- Date & Codd: formal mathematical underpinnings of relational databases
  - Many years to make this work:
    - e.g. query optimization - R join S = S join R
    - transactions - row level locks
  - Centered on syntax, correctness.
    - Why?
    - make it work at all.

Then what happened?

- We had to make it work in the real world

First Step, re: Real World Introducing Primary Index

If primary key is one attribute

```
CREATE TABLE t_test
 (a INTEGER PRIMARY KEY,
  b VARCHAR(10)
 )
```

If primary key is a number of attributes it is called a *composite key*.

```
CREATE TABLE track(
  album CHAR(10),
  dsk INTEGER,
  posn INTEGER,
  song VARCHAR(255),
  PRIMARY KEY (album, dsk, posn)
)
```

Values in the primary key column(s) are unique
➔ easy & fast to locate a particular row

simple key: one attribute

compound key: multiple attributes

Constraint Syntax and Terminology

If primary key is one attribute

```
CREATE TABLE t_test
 (a INTEGER PRIMARY KEY,
  b VARCHAR(10)
 )
```

If primary key is a number of attributes it is called a *composite key*.

```
CREATE TABLE track(
  album CHAR(10),
  dsk INTEGER,
  posn INTEGER,
  song VARCHAR(255),
  PRIMARY KEY (album, dsk, posn)
)
```

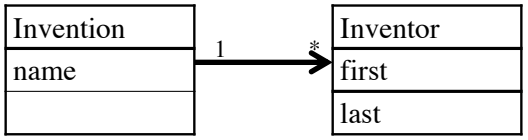
PRIMARY KEY a.k.a. a "primary key constraint"
- if the constraint is on a single attribute --> column constraint
- if the constraint is on a multiple attributes --> table constraint

simple key: one attribute

compound key: multiple attributes

Inventor-Invention, Object Model *UML Class Diagram* Logical Model

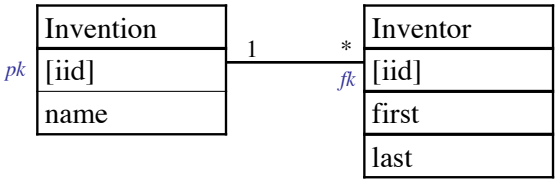
A list of inventions, each with their list of inventors



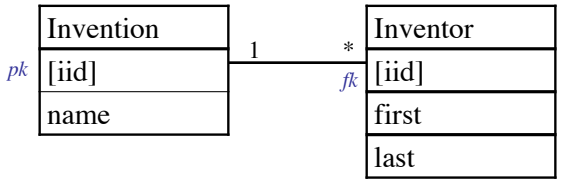
a 1 to many relationship

a *logical* model

Inventor-Invention, Physical Data Model (UML syntax)



Inventor-Invention, Physical Data Model
(UML syntax)



Tool dependent:
- PK, FK, PFK labels on the diagram
- Name munging in the creation of key column names
  - E.g. PK_Invention_ID
  - FK_Invention_PK_Invention_ID

Inventor - Invention as DB Tables

Invention	
iid	name
1	structure
2	sequencing_machine
3	expression_chips

Inventor		
iid	first	last
1	Francis	Crick
1	James	Watson
1	Rosalyn	Franklin
1	Maurice	Wilkens
2	Lee	Hood
3	David	Botstein

Who invented the sequencing machine?

```
Select last
From Invention, Inventor
Where
Invention.name="sequencing_machine"
and
    Invention.iid = Inventor.iid
```

foreign key implements 1 to many *relation*

Of Foreign Keys, Pointers and Surrogates

Surrogate: An instance of a data type that can be used to locate a disk block or a row in a database. Can be a key or row id (RID)

- informally, but *never* formally, a pointer.

The Problem

- Relational representations are semantically challenged.
  - Formal completeness results that show it is possible to compute/represent "rich" concepts.
  - Complicated structures are not *expressed* elementally (usage, like chemistry)
  - As object programming arose, issues became more obvious (painful)

Solution, two parts

- DBMS provides facilities that support richer ideas of database consistency.
  - Constraints, Views (declarative)
  - Triggers (procedural constructs)
- Modeling tools integrate object tools and *compile* the representation to relational form.

The Constraints

https://www.tutorialspoint.com/sql/sql-constraints.htm

- NOT NULL - column cannot have NULL value.
- DEFAULT – Provides a default value for a column when none is specified.
- UNIQUE – all values in a column are different.
- PRIMARY Key – Uniquely identifies each row/record in a database table.
- FOREIGN Key –
- CHECK Constraint – The CHECK constraint ensures that all the values in a column satisfies certain conditions.

Foreign Key: References

- In relation R
  - "attribute A references S (B)"
    - values in the A column of R must
    - uniquely appear
    - in the B column of relation S .
  - (Oracle) B must be declared the primary key for S .

Why declare this to the database?

- declare == (inform and enforce)
- performance
  - join example - on the board
  - correctness

Example

- CREATE TABLE Beers (
  name CHAR(20) PRIMARY KEY,
  manf CHAR(20)
  );
  CREATE TABLE Sells (
  bar CHAR(20),
  beer CHAR(20) REFERENCES Beers(name),
  price REAL
  );

Alternative

- Add another element declaring the foreign key, as:
  CREATE TABLE Sells (
  bar CHAR(20),
  beer CHAR(20),
  price REAL,
  FOREIGN KEY beer REFERENCES
  Beers(name)
  );
- Extra element essential if the foreign key is more than one attribute.

Exceptions

- Two ways:
  - 1. Insert a Sells tuple referring to a nonexistent beer.
    - Always rejected.
  - 2. Delete or update a Beers tuple that has a beer value some Sells tuples refer to.
    - a) Default: reject.
    - b) Cascading delete: Ripple changes to referring Sells tuple.

Example

- Delete "Bud." Cascade deletes all Sells tuples that mention Bud.
- Update "Bud" → "Budweiser." Change all Sells tuples with "Bud" in beer column to be "Budweiser."

What Happens On a Foreign Key Exception

- c) Set Null : Change referring tuples to have NULL in referring components.
- Example
- Delete "Bud." Set-null makes all Sells tuples with "Bud" in the beer component have NULL there.
- Update "Bud" → "Budweiser." Same change.

Selecting a Policy

- Add ON [DELETE, UPDATE] [CASCADE, SET NULL] to declaration of foreign key.

Selecting a Policy (II)

- Example

```
CREATE TABLE Sells (
bar CHAR(20),
beer CHAR(20),
price REAL,
FOREIGN KEY beer REFERENCES
Beers(name)
ON DELETE SET NULL
ON UPDATE CASCADE
);
```


Selecting a Policy (III)

- "Correct" policy is a design decision.
  - E.g., what does it mean if a beer goes away? What if a beer changes its name?

Attribute-Based Checks

- Follow an attribute by a condition that must hold for that attribute in each tuple of its relation.
- Form: CHECK (condition).
  - Condition may involve the checked attribute.
  - Other attributes and relations may be involved, but only in subqueries.
  - Oracle 7.3.2: No subqueries allowed in condition.

Attribute-Based Checks

- Condition is checked only when the associated attribute changes (i.e., an insert or update occurs).

Example

```
CREATE TABLE Sells (
    bar CHAR(20),
    beer CHAR(20) CHECK(
        beer IN (SELECT name
        FROM Beers)              // much like fk constraint
        ),
    price REAL CHECK(
        price <= 5.00
        )
);
```


Attribute-Based Checks (III)

- Check on beer is like a foreign-key constraint, except
  - The check occurs only when we add a tuple or change the beer in an existing tuple, not when we delete a tuple from Beers.

Tuple/Table-Based Checks

- Separate element of table declaration.
- Form: like attribute-based check.
- But condition can refer to any attribute of the relation.
  - Or to other relations/attributes in subqueries.
  - Again: Oracle 7.3.2 forbids the use of subqueries.

Example

- Only Joe's Bar can sell beer for more than $5.

```
CREATE TABLE Sells (
    bar CHAR(20),
    beer CHAR(20),
    price REAL,
    CHECK(bar = 'Joe''s Bar'
          OR price <= 5.00)
);
```


Triggers

- Often called event-condition-action rules.
- *Event* = a class of changes in the DB, e.g., "insert into Beers."
- *Condition* = a test as in a where-clause for whether or not the trigger applies.
- *Action* = one or more SQL statements.

Example

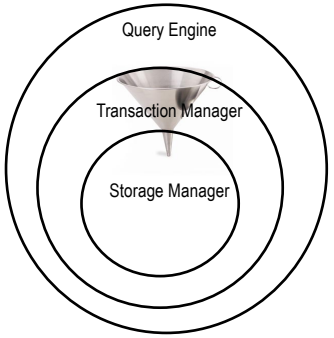
- Whenever we insert a new tuple into Sells, make sure the beer mentioned is also mentioned in Beers, and insert it (with a null manufacturer) if not.

Example (II)

- CREATE TRIGGER BeerTrig
  AFTER INSERT ON Sells
  FOR EACH ROW
  WHEN(new.beer NOT IN (SELECT name FROM Beers))
    BEGIN
    INSERT INTO Beers(name)
    VALUES(:new.beer);
  END;

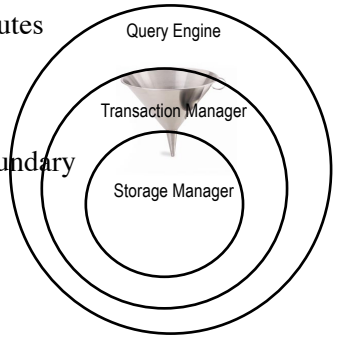
How can the DBMS support this? (internal architecture)

- All updates *funnel* through the transaction manager



Funnel is implemented in the SQL Engine

- SQL interpreter executes
  - insert, delete, update
  - *commit*
- Commit is a clear boundary
  - actions
    - before
    - after



Triggers

- Standardized in SQL3
- Prior to SQL3, vendor specific syntax

  (vestigial compatibility exists)
- Action part, of ECA is a procedural program
  - every vendor has their own procedural extensions. e.g. PSQL, T-SQL,…

Example (II)

- Sells(bar, beer, price)

```
CREATE OR REPLACE TRIGGER BeerTrig
AFTER INSERT ON Sells
FOR EACH ROW
WHEN(new.beer NOT IN (SELECT name FROM Beers))
   BEGIN
   INSERT INTO Beers(name)
   VALUES(:new.beer);
END;
.
Run
```


Options

- Can omit OR REPLACE. Effect is that it is an error if a trigger of this name exists.
- AFTER can be BEFORE.
- INSERT can be DELETE or UPDATE OF < attribute > ON.
- FOR EACH ROW can be omitted, with an important effect: the action is done once for the relation(s) consisting of all changes.

Notes

- There are two special variables new and old, representing the new and old tuple in the change.
  - old makes no sense in an insert, and new makes no sense in a delete.
- Oracle:

in WHEN we use new and old without a colon, but in actions, a preceding colon is needed.

Oracle Notes (II)

- The action is a PL/SQL statement.
  - Simplest form: surround one or more SQL statements with BEGIN and END.
  - However, select-from-where has a limited form.
- Dot and run cause the definition of the trigger to be stored in the database. (stored procedure)
  - Oracle triggers are elements of the database, like tables or views.

Example

- Maintain a list of all the bars that raise their price
  for some beer by more than $1.
  Sells(bar, beer, price)

Example (II)

- CREATE TRIGGER PriceTrig
  AFTER UPDATE OF price ON Sells
  FOR EACH ROW
  WHEN(new.price > old.price + 1.00)
    BEGIN
      INSERT INTO RipoffBars
      VALUES(:new.bar);
    END;
  .
  run