ISTM 6212 - Week 5
RDBMS schema design, joins, etc.

Daniel Chudnov, dchud@gwu.edu

2016-09-27

Agenda

- Project 01 - Reviews
- RDBMS DML: SQL JOINs, subqueries
- RDBMS DDL: schema, normal forms, E-R models
- RDBMS DML: SQL CREATE, INSERT, UPDATE, DELETE
- RDBMS in practice: transactions, functions, triggers
- Exercise 03

Project 01 - Reviews

RDBMS DML - SQL JOINs

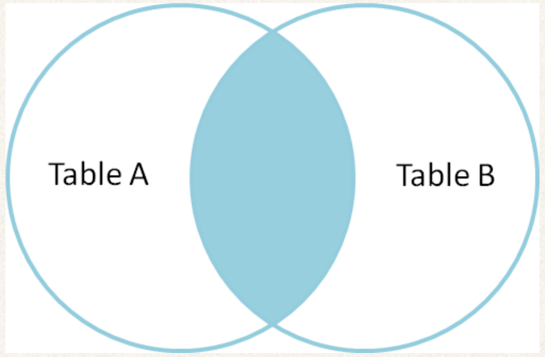
Symbol (Name)	Example of Use
σ (Selection)	$\sigma_{\text{salary>=85000}}(instructor)$
	Return rows of the input relation that satisfy the predicate.
Π (Projection)	$\Pi_{ID,\ salary}(instructor)$
	Output specified attributes from all rows of the input relation. Remove duplicate tuples from the output.
⋈ (Natural Join)	$instructor \bowtie department$
	Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.
× (Cartesian Product)	$instructor \times department$
	Output all pairs of rows from the two input relations (regardless of whether or not they have the same values on common attributes)
∪ (Union)	$\Pi_{name}(instructor) \cup \Pi_{name}(student)$
	Output the union of tuples from the two input relations.

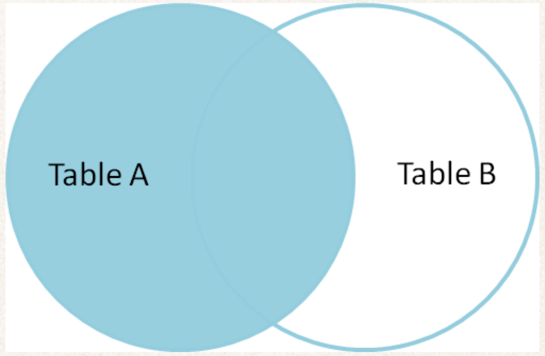
SQL JOIN

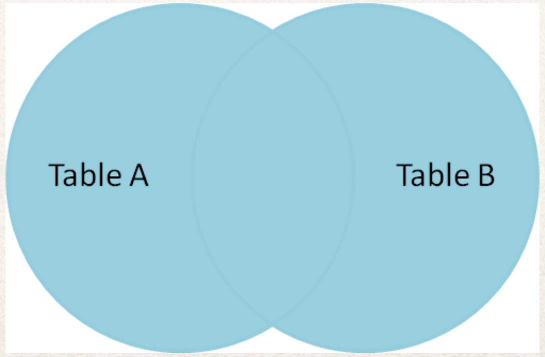
- Cartesian product by default
- further specified by common attributes with ON
- multiple tables, multiple pairs of common attributes
- still all based on set operations

SQL JOIN types

- INNER JOIN
- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN







images from https://blog.codinghorror.com/a-visual-explanation-of-sql-joins/

```sql
SELECT DISTINCT
  display_call_no,
  item_status_desc,
  item_status.item_status,
  perm_location.location_display_name as PermLocation,
  temp_location.location_display_name as TempLocation,
  mfhd_item.item_enum,
  mfhd_item.chron,
  item.item_id,
  item_status_date,
  to_char(CIRC_TRANSACTIONS.CHARGE_DUE_DATE, 'yyyy-mm-dd') AS DUE,
  library.library_display_name,
  holding_location.location_display_name as HoldingLocation
FROM bib_master
JOIN library ON library.library_id = bib_master.library_id
JOIN bib_mfhd ON bib_master.bib_id = bib_mfhd.bib_id
JOIN mfhd_master ON mfhd_master.mfhd_id = bib_mfhd.mfhd_id
JOIN library ON bib_master.library_id = library.library_id
JOIN location holding_location
  ON mfhd_master.location_id = holding_location.location_id
LEFT OUTER JOIN mfhd_item
  ON mfhd_item.mfhd_id = mfhd_master.mfhd_id
LEFT OUTER JOIN item
  ON item.item_id = mfhd_item.item_id
LEFT OUTER JOIN item_status
  ON item_status.item_id = item.item_id
LEFT OUTER JOIN item_status_type
  ON item_status.item_status = item_status_type.item_status_type
LEFT OUTER JOIN location perm_location
  ON perm_location.location_id = item.perm_location
LEFT OUTER JOIN location temp_location
  ON temp_location.location_id = item.temp_location
LEFT OUTER JOIN circ_transactions
  ON item.item_id = circ_transactions.item_id
WHERE bib_master.bib_id = %s
AND mfhd_master.suppress_in_opac != 'Y'
ORDER BY PermLocation, TempLocation, item_status_date desc
```

- if your work requires you to do this often, you will become good at it quickly.
- just think in sets!

RDBMS DML - subqueries

SQL subqueries

- remember: tables are relations and query results are relations
- relations are sets of attribute value instances
- we can use one relation to specify attribute conditions for another

SQL subquery example

- Query 1: SELECT DISTINCT ident FROM person;
- Query 2: SELECT * FROM survey;
    - use Query 1 result within Query 2:

```
SELECT * FROM survey
WHERE person IN
  (SELECT DISTINCT ident FROM person);
```

```python
def get_related_bibids_by_isbn(item):
    if 'isbn' not in item or len(item['isbn']) == 0:
        return []

    binds = ','.join(['%s'] * len(item['isbn']))

    q = '''
    SELECT DISTINCT bib_index.bib_id, bib_text.title
    FROM bib_index, bib_master, bib_text
    WHERE bib_index.bib_id=bib_master.bib_id
    AND bib_master.suppress_in_opac='N'
    AND bib_index.index_code IN ('020N','020A','ISB3','020Z')
    AND bib_index.normal_heading != 'OCOLC'
    AND UPPER(bib_index.display_heading) NOT LIKE '%%%%SET%%%%'
    AND UPPER(bib_index.display_heading) NOT LIKE '%%%%SER%%%%'
    AND bib_text.bib_id = bib_master.bib_id
    AND bib_index.normal_heading IN (
        SELECT bib_index.normal_heading
        FROM bib_index
        WHERE bib_index.index_code IN ('020N','020A','ISB3','020Z')
        AND UPPER(bib_index.display_heading) NOT LIKE '%%%%SET%%%%'
        AND UPPER(bib_index.display_heading) NOT LIKE '%%%%SER%%%%'
        AND bib_id IN (
            SELECT DISTINCT bib_index.bib_id
            FROM bib_index
            WHERE bib_index.index_code IN ('020N','020A','ISB3','020Z')
            AND bib_index.normal_heading IN (%s)
            AND bib_index.normal_heading != 'OCOLC'
            AND UPPER(bib_index.display_heading) NOT LIKE '%%%%SET%%%%'
            AND UPPER(bib_index.display_heading) NOT LIKE '%%%%SER%%%%'
            )
        )
    ORDER BY bib_index.bib_id
```

switch to notebook

RDBMS DDL - schema, E-R models

RDBMS schema / models

- how do we decide which models go where?
- when are sets of attributes one relation, or two, or more?
- design methodologies, normalization

RDBMS schema design methods

- functional analysis of requirements
- clear determination of **cardinality** of relationships:
  - one to one, one to many, many to many
- integration with developer and analyst toolkits

RDBMS schema normalization

- "normal forms" - series of increasingly stringent schema design requirements that build on each other:
  - first normal form (1NF) - atomic values, one table per relation type, primary keys
  - second normal form (2NF) - eliminate dependencies
  - third normal form (3NF) - even fewer dependencies
  - Boyce-Codd normal form (BCNF) - fewer still

RDBMS schema normalization

- great examples in Wikipedia under:
  - https://en.wikipedia.org/wiki/Database_normalization

RDBMS referential integrity

- schema-defined constraints within and between tables must be enforced
- no attribute values outside of acceptable datatype domain, range
- no foreign key references without primary key definitions
- consistent implementation of cardinality rules
- defined handling of cascading updates

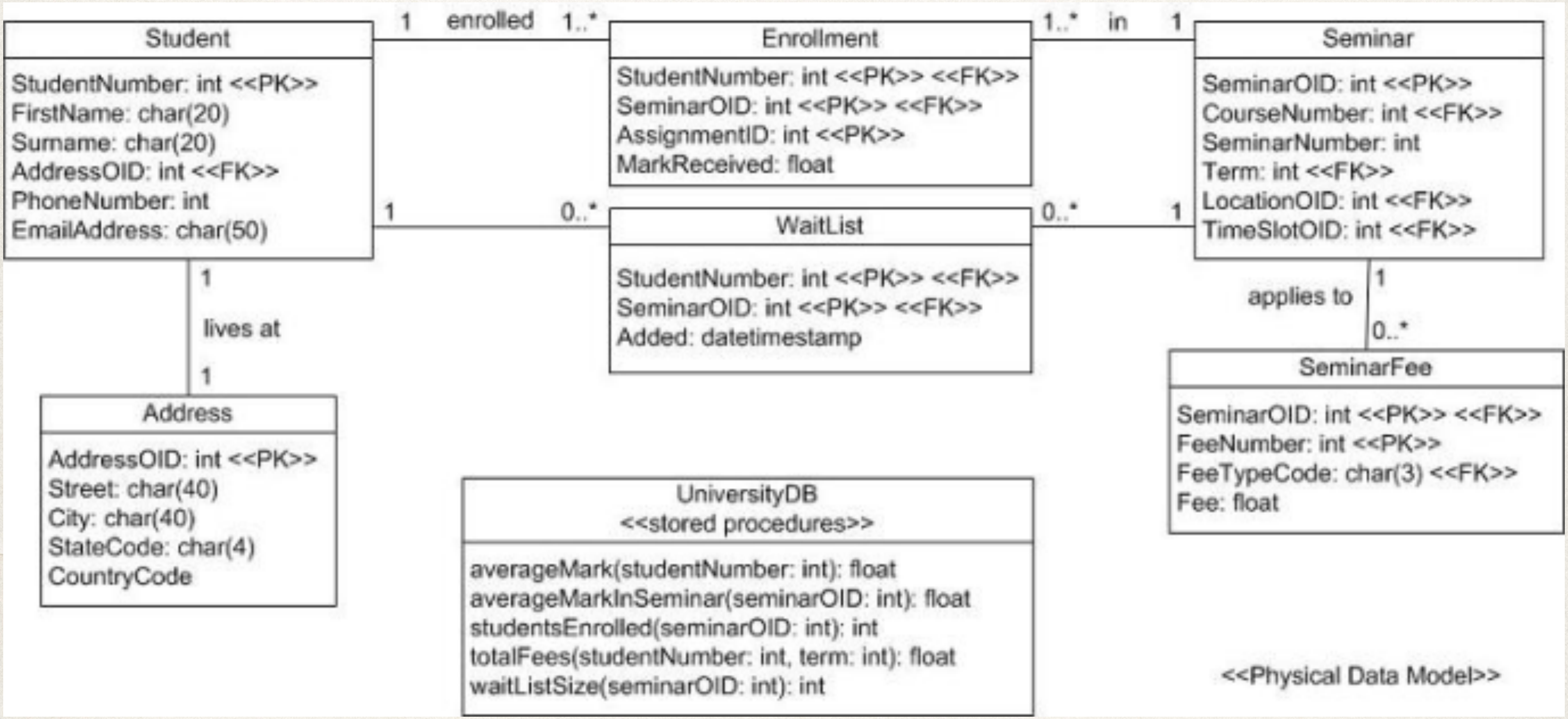
RDBMS normalization - drawbacks

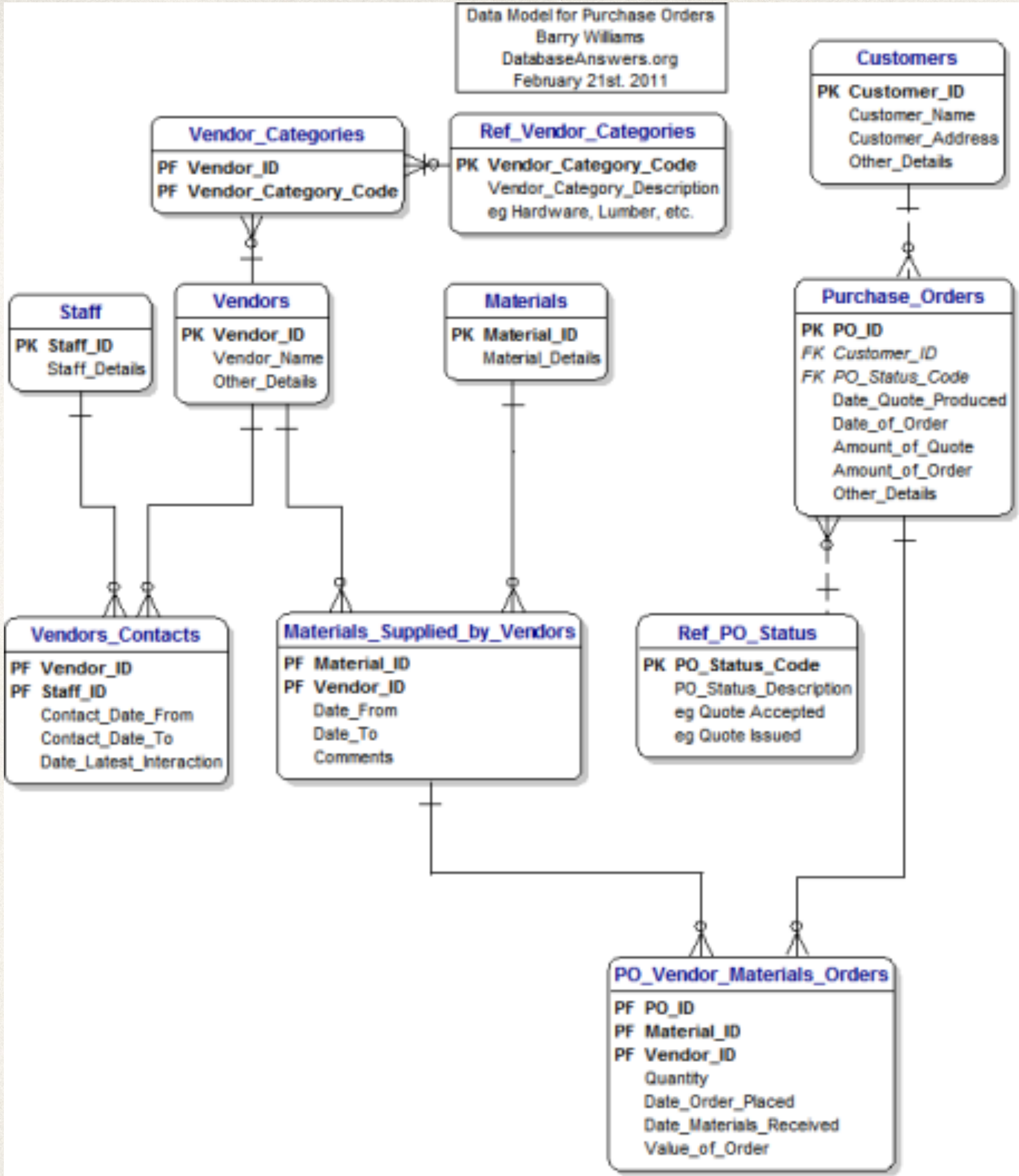
- normalization optimizes for operational applications:
  - storage, **write** (insert/update), integrity efficient
  - scales to very large transaction volumes
  - used everywhere to record business transactions
- lots of work to balance usefulness and soundness
- not always ideal for efficient analysis (**read** heavy)

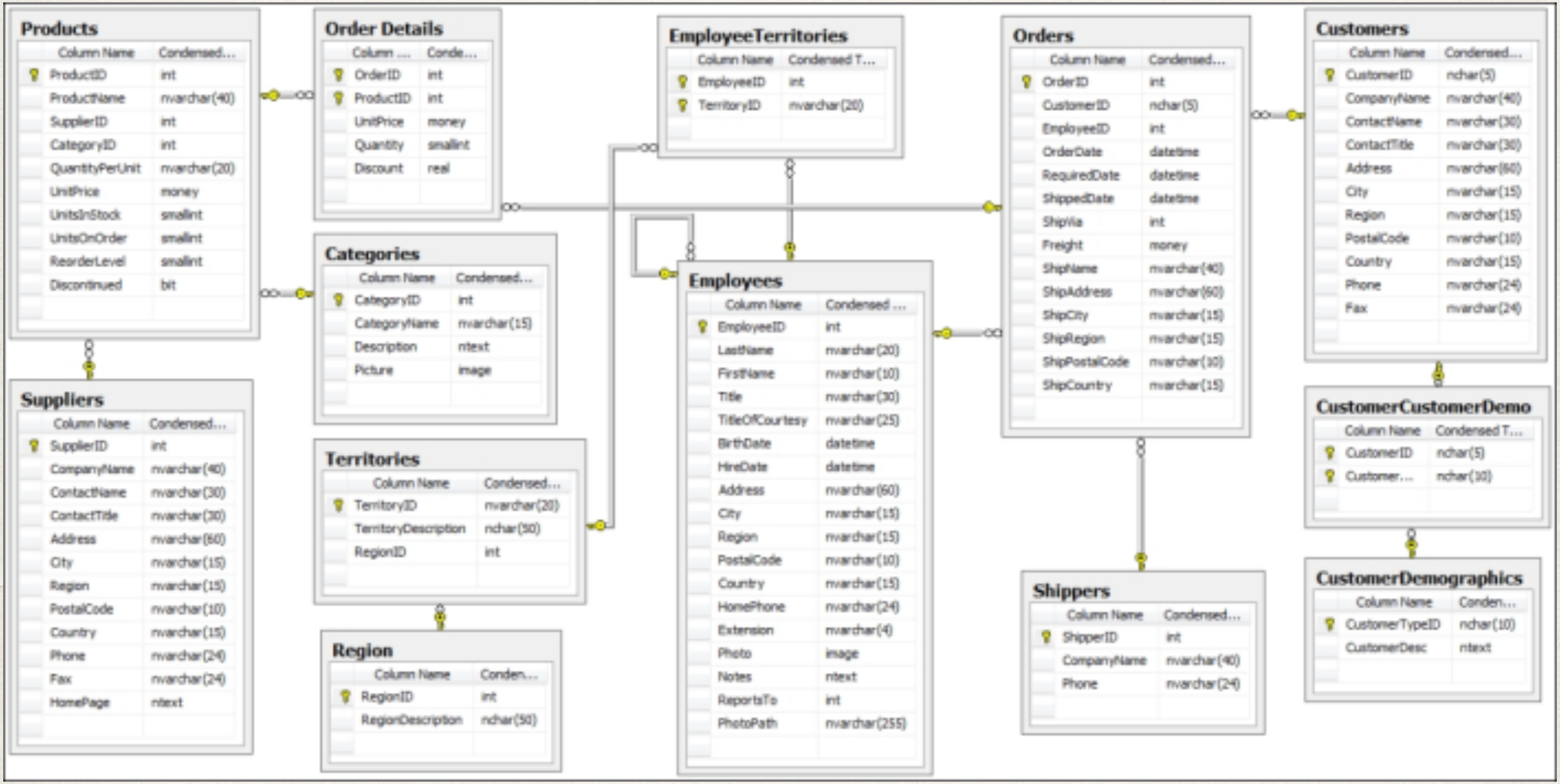
Entity-Relationship models

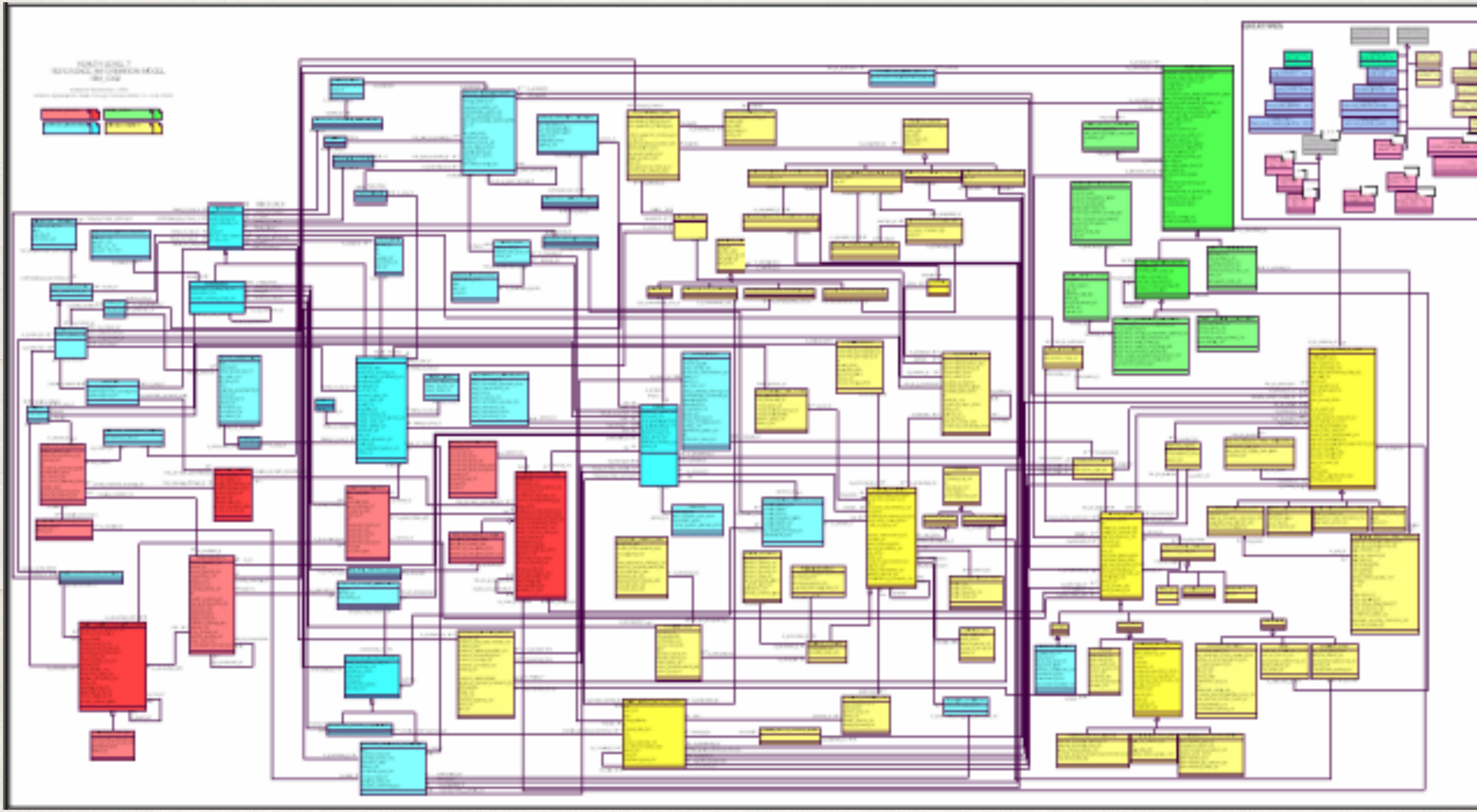
- concise visual form of representing schema
- focus on relation types (tables) and their inter-relationships
- highlight primary keys, foreign keys, cardinality
- ideally include data types, constraints if room allows
- many variations on "visual language" for these

Examples: E-R Diagrams


Student
StudentNumber: int <<PK>>
FirstName: char(20)
Surname: char(20)
AddressOID: int <<FK>>
PhoneNumber: int
EmailAddress: char(50)
1
enrolled
1..*
Enrollment
StudentNumber: int <<PK>> <<FK>>
SeminarOID: int <<PK>> <<FK>>
AssignmentID: int <<PK>>
MarkReceived: float
1..*
in
1
Seminar
SeminarOID: int <<PK>>
CourseNumber: int <<FK>>
SeminarNumber: int
Term: int <<FK>>
LocationOID: int <<FK>>
TimeSlotOID: int <<FK>>
1
0..*
WaitList
StudentNumber: int <<PK>> <<FK>>
SeminarOID: int <<PK>> <<FK>>
Added: datetimestamp
0..*
1
1
lives at
1
Address
AddressOID: int <<PK>>
Street: char(40)
City: char(40)
StateCode: char(4)
CountryCode
1
applies to
0..*
SeminarFee
SeminarOID: int <<PK>> <<FK>>
FeeNumber: int <<PK>>
FeeTypeCode: char(3) <<FK>>
Fee: float
UniversityDB
<<stored procedures>>
averageMark(studentNumber: int): float
averageMarkInSeminar(seminarOID: int): float
studentsEnrolled(seminarOID: int): int
totalFees(studentNumber: int, term: int): float
waitListSize(seminarOID: int): int
<<Physical Data Model>>


Data Model for Purchase Orders
Barry Williams
DatabaseAnswers.org
February 21st. 2011
Customers
PK Customer_ID
Customer_Name
Customer_Address
Other_Details
Vendor_Categories
PF Vendor_ID
PF Vendor_Category_Code
Ref_Vendor_Categories
PK Vendor_Category_Code
Vendor_Category_Description
eg Hardware, Lumber, etc.
Staff
PK Staff_ID
Staff_Details
Vendors
PK Vendor_ID
Vendor_Name
Other_Details
Materials
PK Material_ID
Material_Details
Purchase_Orders
PK PO_ID
FK Customer_ID
FK PO_Status_Code
Date_Quote_Produced
Date_of_Order
Amount_of_Quote
Amount_of_Order
Other_Details
Vendors_Contacts
PF Vendor_ID
PF Staff_ID
Contact_Date_From
Contact_Date_To
Date_Latest_Interaction
Materials_Supplied_by_Vendors
PF Material_ID
PF Vendor_ID
Date_From
Date_To
Comments
Ref_PO_Status
PK PO_Status_Code
PO_Status_Description
eg Quote Accepted
eg Quote Issued
PO_Vendor_Materials_Orders
PF PO_ID
PF Material_ID
PF Vendor_ID
Quantity
Date_Order_Placed
Date_Materials_Received
Value_of_Order


Products
Column Name
Condensed...
ProductID
int
ProductName
nvarchar(40)
SupplierID
int
CategoryID
int
QuantityPerUnit
nvarchar(20)
UnitPrice
money
UnitsInStock
smallint
UnitsOnOrder
smallint
ReorderLevel
smallint
Discontinued
bit
Order Details
OrderID
int
ProductID
int
UnitPrice
money
Quantity
smallint
Discount
real
EmployeeTerritories
EmployeeID
int
TerritoryID
nvarchar(20)
Orders
OrderID
int
CustomerID
nchar(5)
EmployeeID
int
OrderDate
datetime
RequiredDate
datetime
ShippedDate
datetime
ShipVia
int
Freight
money
ShipName
nvarchar(40)
ShipAddress
nvarchar(60)
ShipCity
nvarchar(15)
ShipRegion
nvarchar(15)
ShipPostalCode
nvarchar(10)
ShipCountry
nvarchar(15)
Customers
CustomerID
nchar(5)
CompanyName
nvarchar(40)
ContactName
nvarchar(30)
ContactTitle
nvarchar(30)
Address
nvarchar(60)
City
nvarchar(15)
Region
nvarchar(15)
PostalCode
nvarchar(10)
Country
nvarchar(15)
Phone
nvarchar(24)
Fax
nvarchar(24)
Categories
CategoryID
int
CategoryName
nvarchar(15)
Description
ntext
Picture
image
Employees
EmployeeID
int
LastName
nvarchar(20)
FirstName
nvarchar(10)
Title
nvarchar(30)
TitleOfCourtesy
nvarchar(25)
BirthDate
datetime
HireDate
datetime
Address
nvarchar(60)
City
nvarchar(15)
Region
nvarchar(15)
PostalCode
nvarchar(10)
Country
nvarchar(15)
HomePhone
nvarchar(24)
Extension
nvarchar(4)
Photo
image
Notes
ntext
ReportsTo
int
PhotoPath
nvarchar(255)
Suppliers
SupplierID
int
CompanyName
nvarchar(40)
ContactName
nvarchar(30)
ContactTitle
nvarchar(30)
Address
nvarchar(60)
City
nvarchar(15)
Region
nvarchar(15)
PostalCode
nvarchar(10)
Country
nvarchar(15)
Phone
nvarchar(24)
Fax
nvarchar(24)
HomePage
ntext
Territories
TerritoryID
nvarchar(20)
TerritoryDescription
nchar(50)
RegionID
int
Region
RegionID
int
RegionDescription
nchar(50)
Shippers
ShipperID
int
CompanyName
nvarchar(40)
Phone
nvarchar(24)
CustomerCustomerDemo
CustomerID
nchar(5)
nchar(10)
CustomerDemographics
CustomerTypeID
nchar(10)
CustomerDesc
ntext



very common to see full designs like this

RDBMS DDL - SQL CREATE

SQL CREATE

- define new relation (table)
- simple syntax:
  - table name, attributes, types, constraints
- must be logical (order matters)

SQL CREATE examples

- CREATE TABLE Person(ident TEXT, personal TEXT, family TEXT);
- CREATE TABLE Site(name TEXT, lat REAL, long REAL);
- CREATE TABLE Visited(ident INTEGER, site TEXT, dated TEXT);
- CREATE TABLE Survey(taken INTEGER, person TEXT, quant REAL, reading REAL);

SQL CREATE - examples (2)

```
CREATE TABLE Survey(
    taken   INTEGER NOT NULL, -- where reading taken
    person  TEXT,             -- may not know who took it
    quant   REAL NOT NULL,    -- the quantity measured
    reading REAL NOT NULL,    -- the actual reading
    PRIMARY KEY (taken, quant),
    FOREIGN KEY (taken) REFERENCES Visited(ident),
    FOREIGN KEY (person) REFERENCES Person(ident)
);
```

diagram vs. schema vs. model

- E-R diagrams provide high-level overviews but might lack specifics
- DDL schema code provides low-level details but lacks high-level summary
- both are necessary in all but trivial databases

RDBMS DML -
SQL INSERT, UPDATE, DELETE

SQL INSERT

- add rows to a table
- attributes must align (explicitly or implicitly)
- must abide by table definition
- may insert many rows at once

SQL INSERT examples

- CREATE TABLE Site(name TEXT, lat REAL, long REAL);
- INSERT INTO Site VALUES ('DR-1', -49.85, -128.57);
- INSERT INTO Site VALUES ('DR-3', -47.15, -126.72);
- INSERT INTO Site VALUES ('MSK-4', -48.87, -123.40);
- INSERT INTO Site (lat, long, name) VALUES (-49.85, -128.57, 'DR-1'), (-47.15, -126.72, 'DR-3'), (-48.87, -123.40, 'MSK-4');

SQL INSERT examples (2)

- CREATE TABLE JustLatLong(lat text, long text);
- INSERT INTO JustLatLong SELECT lat, long FROM Site;

SQL UPDATE

- change existing records
- won't add new or delete existing records
- must abide by schema constraints
- can use subqueries to extract or constrain values from data located elsewhere
- easy to make mistakes!

SQL UPDATE - examples

- compare:
  - UPDATE Site SET lat = -48.87, long = -125.40;
  - UPDATE Site SET lat = -48.87, long = -125.40 WHERE name = 'MSK-4';
- what effects can forgetting a constraint have?

SQL DELETE

- remove existing records
- must abide by integrity constraints
- can use subqueries to extract or constrain values from data located elsewhere
- just as easy to make mistakes!

SQL DELETE - examples

- compare:
  - DELETE FROM Site;
  - DELETE FROM Site WHERE name = 'MSK-4';
- what effects can forgetting a constraint have?

switch to notebook

RDBMS in practice: transactions, functions, triggers

Transactions

- "ACID" properties:
  - Atomicity - all steps complete or all fail
  - Consistency - db is consistent whether transaction succeeds or fails
  - Isolation - two simultaneous transactions follow rules for each to complete independently
  - Durability - survive system failure, etc.

Transactions - example

```
START TRANSACTION;
  UPDATE account SET balance=balance-900
    WHERE account_num=9001;
  UPDATE account SET balance=balance+900
    WHERE account_num=9002;
  COMMIT;
ROLLBACK;
```

Functions

- allows users to define commonly used operations as functions for use in SQL statements
- similar to function definition in procedural programming
- defines name, parameters, return type
- often requires additional permissions

Functions - example

```
CREATE FUNCTION CtoF(Celsius FLOAT)
  RETURNS FLOAT
  RETURN (Celsius * 1.8) + 32;

SELECT name, CtoF(boiling_point)
FROM elements;
```

Triggers

- based on some conditions, perform specific operations
- INSERTs, UPDATEs, and DELETEs are typical trigger actions
- often used for formal logging of activities and data changes

Triggers - example

```
CREATE TRIGGER Books_Delete
AFTER DELETE ON Books
  REFERENCING OLD ROW AS Old
FOR EACH ROW
  INSERT INTO Books_Deleted_Log
      VALUES (Old.title);
```

Exercise 03