Physical Design

Physical Database Design (Defined): Process of producing a description of the implementation of the database on secondary storage; it describes the base relations, file organizations, and indexes used to achieve efficient access to the data, and any associated integrity constraints and security measures.

What does that mean for us? We will describe the plan for how to build the tables, including appropriate data types, field sizes, attribute domains, and indexes. The plan should have enough detail that if someone else were to use the plan to build a database, the database they build is the same as the one you are intending to create.

The conceptual design and logical design were independent of physical considerations. Now, we not only know that we want a relational model, we have selected a database management system (DBMS) such as Access or Oracle, and we focus on those physical considerations.

Logical vs. Physical Design: Logical database design is concerned with ***what*** to store; physical database design is concerned with ***how*** to store it.

Although we have discussed user requirements, conceptual design, logical design, and physical design as if they are sequential activities, they do not occur in isolation. In reality there is often feedback between physical design, conceptual/logical design, and application design…all of which are aimed at meeting the user requirements. For example, decisions made during physical design to improve performance may include merging tables together – which in turn should be reflected in the logical model and will have an effect on the application design.

Meeting the needs of the users is the gold standard against which we measure our "success" in creating a database.

OUR NEW CHIP IS SLOWER THAN OUR COMPETITION'S PRODUCTS.
scottadams@aol.com
www.dilbert.com
WE'LL CLAIM WE'RE THE FASTEST. IF ANYONE DOES BENCHMARK TESTS, WE'LL SAY THEY USED OLD DRIVERS.
©2004 Scott Adams, Inc./Dist. by UFS, Inc.
WHENEVER I TALK TO YOU, I FEEL LIKE I SHOULD BE WEARING A WIRE.
SINCE WHEN IS MARKETING A CRIME?
© UFS, Inc.

Underlying Concepts

Because physical design is related to how data are physically stored, we need to consider a few underlying concepts about physical storage. One goal of physical design is optimal performance and storage space utilization. Physical design includes data structures and file organization, keeping in mind that the database software will communicate with your computer's operating system. Typical concerns include:

- Storage allocations for data and indexes
- Record descriptions and stored sizes of the actual data
- Record placement
- Data compression, encryption

DB-related physical components & concepts:

- Kernal: software internal to the DBMS
  - Controls storage, retrieval, security
  - Interacts with "host services" (functions provided by the operating system, like how to access your CPU, how to use your computer's RAM…)
  - Data Dictionary: defines data structures used by the kernal (the data dictionary contains the database meta-data). There is an external data dictionary (which we will write) and an internal one that is maintained by the DBMS when we create DB structures.
  - Database Engine: software that reads the data dictionary, accesses data.

Physical Design Steps (Methodology)

The process of describing the DB implementation on secondary storage can include all the following. We will not practice all of the steps in our final project**.**

- Involve the users! (Throughout the process)
- Translate logical data model for target DBMS
  - Design base relations (tables)
  - Determine how to represent derived data
  - Design general constraints (methods to enforce business rules)

For the final project: We will create a data dictionary

- Design physical representation – file organization and indexes
  - Analyze transactions
  - Choose file organizations
  - Choose indexes
  - Estimate disk space requirements

For the final project: We will identify one index in addition to the PK

- Design user views
- Design security mechanisms
- Consider the need to introduce controlled redundancy (denormalization)
- Monitor and tune operational system *

We will discuss these briefly but not implement them.

* not really design – done after the system is built. But typically a PLAN for this is developed.

Detail of Physical Design Methodology

User involvement

Users should be involved in ALL phases of the database development process. We clearly needed users to define the system requirements. We went back to the users to clarify the entities, relationships, and cardinality when creating the ERD. Occasionally we need further clarification from users during logical design and normalization (Will you ever sort by name [Do I need to split name into last and first]?).

Similarly, we may need to back to the users during physical design, especially to clarify enterprise constraints (business rules). We need to consider the format of data in the system. For example, data that appear to be numbers might not be best stored as numbers. ICD-9 (or ICD-10) codes often contain a 0 at the beginning or the end of the code, and that 0 is meaningful. 123.0 is different than 123 and 012.1 is different than 12.1 *{not real ICD codes!}*. Similarly, consider social security numbers, zip codes, student ID numbers…leading or trailing 0 may be important in those as well, and those leading/trailing 0 would be LOST if we stored this as a "number" data type. Such data are best stored as text even though they LOOK like numbers. My rule of thumb is to ask myself if I would be doing math with the number. If not, consider whether it needs to be stored as text data.

Translate Logical Data Model

So…how do we go about selecting the DBMS?

- Consider potential size of the DB, number of users, type of interfaces. Access is readily available as part of MS office, for example, but it is "Windows only" – there is no "native" MAC version of MS Access. Mac users can sometimes work around this, for example, by using parallels or by logging in to a virtual PC such as a citrix/metaframe client. But it's something to consider.
- Consider cost. There are some nice, free, open-source databases available
- Consider trade-offs (e.g., ease of use or cost vs. lack of features). Many open-source/free DB come with little or no support or documentation, or may lack certain desired features.
- Consider organizational constraints (e.g., your organization may mandate certain DBMS)

We need to know the functionality of the target DBMS. Considerations include HOW to create tables, how the system defines and deals with keys (not only the primary key [PK], but also foreign keys [FK] and alternate/candidate keys [AK]), whether the system supports "required" (not null) constraints. Other considerations or whether the system supports: definition of domains ("allowed values"), enforcement of relational integrity constraints, and enforcement of other constraints.

An important consideration is: What are the data types allowed in the DBMS?
(We will explore Access and Oracle data types soon)

Design Tables (the book calls this "design base relations")

Decide how to represent the data tables (called base relations in the formal definition) in the target DBMS.

From the logical design, we have a set of table schemas that describe

- the name of the table
- the name of stored attributes (the "columns").
- the PK and, where appropriate, AK and FK

We also might have

- referential integrity constraints for FKs (what table does the foreign key link to)
- assumptions

Now we'll add additional information. For each attribute, we will specify:

- domain: consisting of data type, length, and constraints on the domain (we'll talk more about constraints soon)
- whether the attribute can hold nulls
- default value for the attribute (optional)
- any derived attributes and how they should be computed

We will create a data dictionary for the final project. More about this later in the module, but a large component of the data dictionary is the table design. I like to organize this into Word or Excel tables, like shown below for the Demog table in the SNDB, built in MS Access

Demog Table

Attribute Name	**Description**	**Data Type**	**Data Length**	**Domain**	**Allow Null?**
pt_id	Patient identifier.	Text	4	Any positive integer	No (PK)
name	Patient name	Text	20	Formatted as last name, first name	Yes
zip	Patient's zip code	Text	5	99999	Yes
Gender	Patient gender	Text	1	Coded in the listgender table	Yes
Race	Patient race	Text	1	Coded.	Yes

Primary Key: pt_id

Notes about the example

- You could make the last column "required" instead of "allow null?", in which case each value would be exactly reversed. The primary key field is always required—if the PK is multiple columns, all the PK columns are required.
- You can indicate the primary key within the table, or below the table. You don't need to do both. If the description is obvious from the attribute name, you can leave that blank. In many cases, though, a description is helpful.
- Attribute name is same meaning as variable name or column name. You can use any of these "names" for that first column.
- Domain can be descriptive ("any positive integer"), a pattern or example (99999), a list of specific values, reference to a list of values, etc.

PHYSICAL DESIGN IS DBMS SPECIFIC -- Naming

Note that I specified the table design above is applicable to MS ACCESS. Access is very “forgiving” when it comes to things like names. You can have spaces, you can pretty much do what you want.

Other DBMS, such as Oracle, are more strict. Most DBMS, in fact, have naming restrictions similar to those in Oracle. Attribute names must begin with a letter, you cannot have spaces (although you CAN have underscore). You can’t use “reserved words” such as the name of functions (max) or datatypes (date).

So one step you might need to take at this point is to **check your table and attribute names**. An easy way to fix issues with reserved words is to append some meaningful prefix or suffix, like changing *date* to *test_date* or *visit_date*. Sometimes you can just make a name plural, for example, Oracle does not allow *comment* as a column name, but will allow *comments*.

PHYSICAL DESIGN IS DBMS SPECIFIC – Data Types

Although there is a reasonably small set of primitive data types (numbers, letters, images, etc), the way that different DBMS deal with those data types, the names they use for the data types, and constraints regarding the data types vary.

Alphanumeric data

This is a generic data type that includes letters, numbers, symbols – the sort of “characters” we are used to working with in a word processor.

Oracle calls this type of data CHAR for fixed length fields (always stores the full length allowed – anything not filled in with “text” will be filled in with spaces); it uses VARCHAR2 for variable length fields (stores the number of characters typed). There is a VARCHAR data type for variable length fields as well, but that is likely to be deprecated in future releases, so VARCHAR2 is preferred over VARCHAR.

MS Access calls this type of field TEXT. It is always variable length – there is no fixed length character data type in Access.

Other systems may call this type of field “string”, “character” or other names.

With both Oracle and Access, you must specify a field length. This is the total field length for CHAR (Oracle), and is the **MAXIMUM field length** for VARCHAR, VARCHAR2 (Oracle) and TEXT (Access). In determining DB size, each character is 1 byte. (Unless you are using a 2-byte character data set like Chinese, which requires a specific setting on your operating system and within the DBMS).

So VARCHAR2(25) [Oracle] or TEXT(25) [Access] is an alphanumeric field that can hold UP TO 25 characters, and will take up to 25 bytes of space per row.

Numeric data (numbers)

Oracle uses number (p,s) format for numbers; where p is the precision (how many digits total) and s is the scale (how many of the p digits are after the decimal point). So a number like 9.25 would be represented as number (3,2) – there are 3 digits, 2 of which are after the decimal. Some versions of Oracle allow you to specify numbers by type, such as integer, long, etc but that practice is discouraged for two reasons: (1) those typed numbers take the maximum allowable space for the type – usually more space than you need, and (2) the functionality may go away in future versions.

Access uses number types, though--integer or long integer for whole numbers, and single or double for numbers with a decimal component. Access also has a *currency* data type (all numbers and math done with the number use only 2 digits, like when computing with money).

Access has an "autonumber" data type, which is a LONG INTEGER that is automatically generated by the DB. You can have the system increment (count up sequentially) or generate a pseudo-random number. *I say "pseudo" random because it's really based on a computation that is applied to the time on the system clock.*

Oracle is trickier for auto-generated numbers. You need to create a separate "counter" table, and retrieve the number from that table with a trigger that pulls the number and then increments the counter (triggers are code that run automatically).

For a nice description of Oracle data types, including the size associated with different data types, see http://www.ss64.com/orasyntax/datatypes.html

For a nice description of MS Access data types, see http://www.databasedev.co.uk/fields_datatypes.html

Constraints: Limits on the data
Keeping in mind enterprise constraints (business rules), determine:

- required data – i.e. NOT NULL
- relational integrity constraints (referential integrity)
- domain for the data.

The domain includes **physical aspects** defined with the attribute (data type, field size). It also includes **constraints imposed by the business rules**. Some of these constraints can be implemented with physical measures within the database, others might be managed by the application (user interface), still others might be managed through policy and procedures. The domain includes ALL of these – physical constraints, application management, and policy/procedural constraints.

Let's look at some of the physical measures we can use to enforce constraints.

- Not Null: The field is never allowed to be empty. Data must be entered at the time the row is created. This is not the same as data that are needed eventually--requiring that data might be a policy measure. Keep in mind that if you specify "not null" as a constraint, you cannot save the row of data if that column is blank.

- Unique: No duplicate data when you look across rows. A unique constraint can be used for a candidate (alternate) keys (*Access calls this the "no duplicates" property).*

- Primary Key: Adding the primary key constraint automatically includes **not null** constraint + **unique** constraint (+ an index).

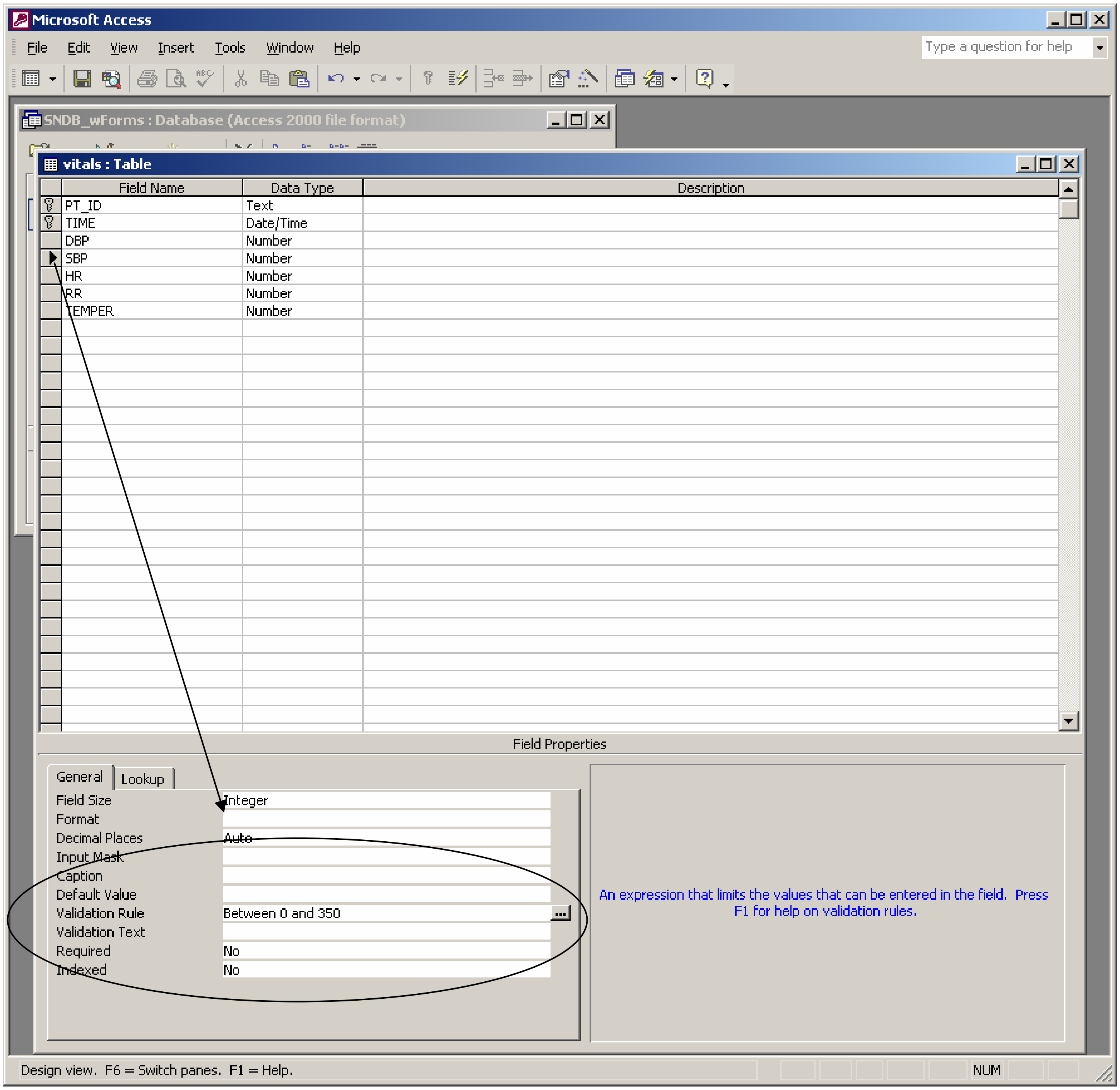
- Foreign Key: using the foreign key constraint enforces referential integrity. Every foreign key must match a primary key or a unique constraint on another table. The sequence of data entry is affected here - you must enter data into the “main” table before it can be used as a foreign key. *(Access uses properties within the “relationships” window to enforce referential integrity. Oracle uses “references” statement within the table constraints).*

- Check. A check constraint helps to enforce attribute domains.
  - Boolean logic (“Check SBP >=0 and SBP <=350” or “Check SBP Between 0 and 350”). This would be evaluated as true or false; if false you get an error message and can't store the row of data.
  - The **check constraint** is called a **validation rule** in Access.
  - *The above rule would also act like a NOT NULL statement. Why? Because null data are not between 0 and 350. How would you correct this to allow null values? By adding "...or is null"*

  Limits on check constraint: Can’t use a comparison (such as percent of the total) or a column that varies over time (such as sysdate [the current date on your operating system clock)]. Use triggers, instead, if you want to enforce this sort of cross field or complex check.

Oracle example – when you create the table, you could list the check constraint in the create table statement:
Create table vitals (…[list of attributes and data types],
SBP Number (3) check SBP between 0 and 350, …

In Access you would create the SBP column in table design view, then in the properties for this column, create the validation rule:



Field Name	Data Type	Description
PT_ID	Text	
TIME	Date/Time	
DBP	Number	
SBP	Number	
HR	Number	
RR	Number	
TEMPER	Number	

Field Size	Integer
Format	
Decimal Places	Auto
Input Mask	
Caption	
Default Value	
Validation Rule	Between 0 and 350
Validation Text	
Required	No
Indexed	No

An expression that limits the values that can be entered in the field. Press F1 for help on validation rules.

Some DBMS provide more facilities than others for defining enterprise constraints. An example of a complex constraint that <u>might</u> work in Oracle, but that <u>would not</u> work in Access:

```
        CONSTRAINT StaffNotHandlingTooMuch
                CHECK (NOT EXISTS (SELECT staffNo
                                        FROM PropertyForRent
                                        GROUP BY staffNo
                                        HAVING COUNT(*) > 100))
```

Triggers: These are stored procedures (code) that “fire” automatically when data are manipulated-when there is an insert, update, or delete statement. Triggers can call (use) other code modules. Constraints are faster than triggers but triggers can be more complex.

In Oracle, triggers are created within PL*SQL (the internal Oracle programming language).

Access does not allow triggers on tables. If you use forms, you can emulate the action of triggers by writing code inside the form.

More about domain constraints

Domain constraints are limits on the data values. What business rules can we implement in some physical manner? For example, suppose our business rule says that a valid SBP must be between 0 and 350. We can do this with a check constraint (Oracle) or validation rule (Access) as discussed above.

You might enforce business rules in the user interface instead of at the table level: e.g., use a picklist of values and restrict users to only choosing something from the picklist. This would work nicely when the list of choices is known. For very small lists, you might consider coding the data numerically, and using radio buttons, like:

Gender: ○ Male ○ Female   where Male = 1 and Female = 2

*radio buttons can only store NUMERIC data. Each choice is given a number code, and the user is allowed to choose only ONE of the choices – like the channel selection buttons on a car radio.

Check boxes only store true/false or yes/no values (sometimes called Boolean data). In Access there is a yes/no data type. Yes (or “true”) is stored as -1 and no (or “false”) is stored as 0. Most versions of Oracle do not allow a Boolean data type – you would use a number field of size 1 and store the values, but you’d have to know what the numbers mean. A few DBMS allow a three-valued “boolean like” field, equating to yes, no, and unknown.

So radio buttons are an option for “pick ONE of the following”. Check boxes typically imply that **any or all** of the boxes may be chosen – you store a “yes” or “no” for each choice.

See how the application interface design might influence some of your physical design decisions?

Some business rules can only be enforced through policy/procedure. Consider the rule “FirstName must be a valid first name.” This must be enforced through policy/procedure. With physical constraints, we can only specify that this is text, and the length of the text, but we can’t determine if the text is a valid first name.

Derived Data

Examine the user requirements, ERD, logical data model and data dictionary, and produce a list of derived attributes. Derived attribute can be stored in database or calculated every time they are needed. Document what is to be done! Your choice may be based on: space to store the derived data, effort to keep it consistent with the data from which it is derived; versus cost (query time) to calculate the value each time it is required. Less expensive option might be chosen subject to performance constraints.

Code Tables (Lookup tables)

Code tables list a code and the meaning of the code, like:

ShipCode	**Shipper**
1	UPS
2	FedEx
3	USPS ground

In a main data table, you would store the CODE. Then you can look up the value.

The SNDB listGender table is another example.

When we build the interface for out DB, we can use code tables or "look up" tables (single column list of choices) to populate list boxes and combo boxes. This can simplify data entry for your users – those interface boxes can be set to display the meaning to the user, but store the code in the table.

In your final project I ask you to create a code table for ONE column somewhere in the DB. Include that code table in your data dictionary, and list it in the domain column of where it will be used. See the "gender" description in the Demog table on the next page.

At this point, you can **BEGIN your final project data dictionary**. For each table, include

- Table name
- Attribute names
- Description of the attribute if not apparent from the name
- Data type – must be consistent with the DBMS you will use to build the database
- Length of the field. Use the data type links above to see the choices for length
- Indicate the Primary Key
- If appropriate, also list alt key(s) and foreign key(s)
- Derived attributes and how to compute

You can do this in Word, with a separate Word table for each DB table, like the example below or similar appropriate format

Demog Table

Attribute Name	**Description**	**Data Type**	**Data Length**	**Domain**	**Allow Null?**
pt_id	Patient identifier.	Text	4	Any positive integer	No (PK)
name	Patient name	Text	20	Formatted as last name, first name	Yes
zip	Patient's zip code	Text	5	99999	Yes
Gender	Patient gender	Text	1	Coded in the listgender table	Yes
Race	Patient race	Text	1	Coded.	Yes

Primary Key: pt_id

Or you can do this in Excel, something like:

Table	**Attribute Name**	**Description**	**Data Type**	**Data Length**	**Domain**	**Allow Null?**	**Key/ Index**
Demog	pt_id	Patient identifier.	Text	4	Any positive integer	No	PK
	name	Patient name	Text	20	Formatted as last name, first name	Yes	
	zip	Patient's zip code	Text	5	99999	Yes	
	Gender	Patient gender	Text	1	Coded in the listgender table	Yes	
	Race	Patient race	Text	1	Coded.	Yes	
Adm_dx	pt_id		Text	4	matches demog table	No	PK
	diagnosis	admitting diagnosis	Text	60	medical diagnosis	No	PK