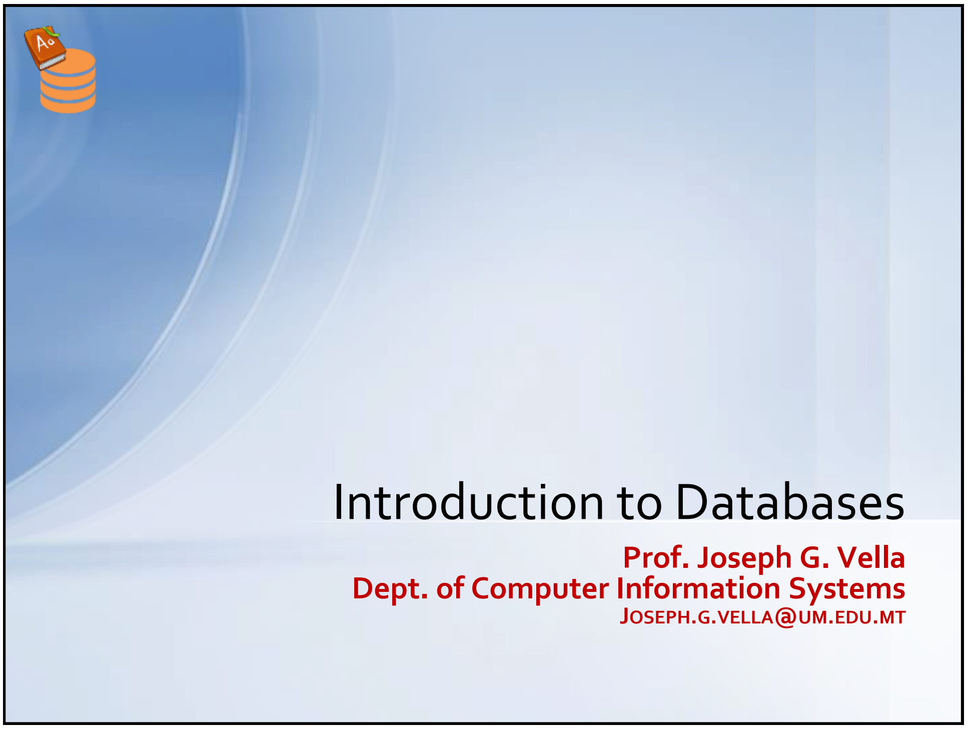
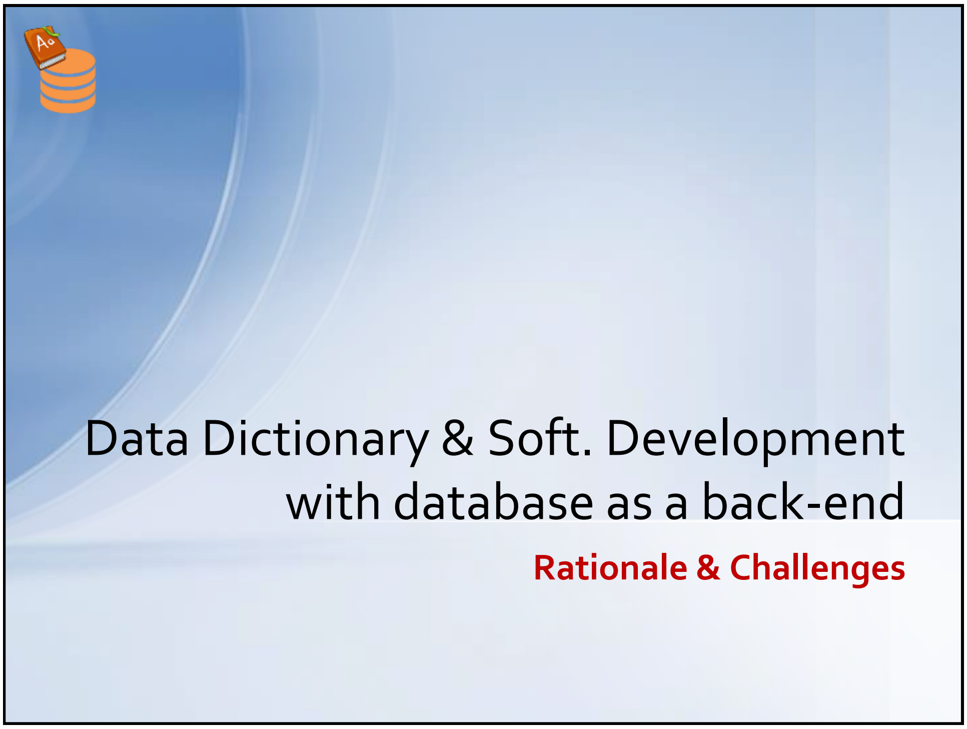


Introduction to Databases

Prof. Joseph G. Vella
Dept. of Computer Information Systems
JOSEPH.G.VELLA@UM.EDU.MT

1



Data Dictionary & Soft. Development with database as a back-end

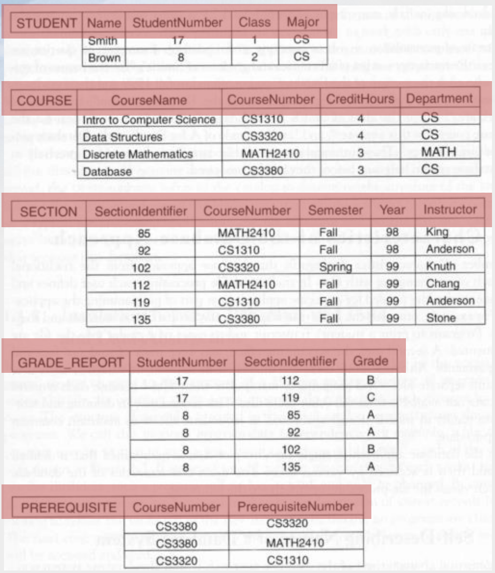
Rationale & Challenges

2

Where had we started …

- **Definition:**
  - A **database** is a collection of structured data stored on persistent storage.
- **Consider the booth strapping problem of the DBMS**
  - Which comes first a database definition or a DBMS?
    - **The data dictionary solution.**
- **As for software development the progress has to be driven by:**
  - greater efficiency in coding effort;
  - increased customization,
  - Increased portability;
  - and ease-of-coding & maintaining.

Sample Database



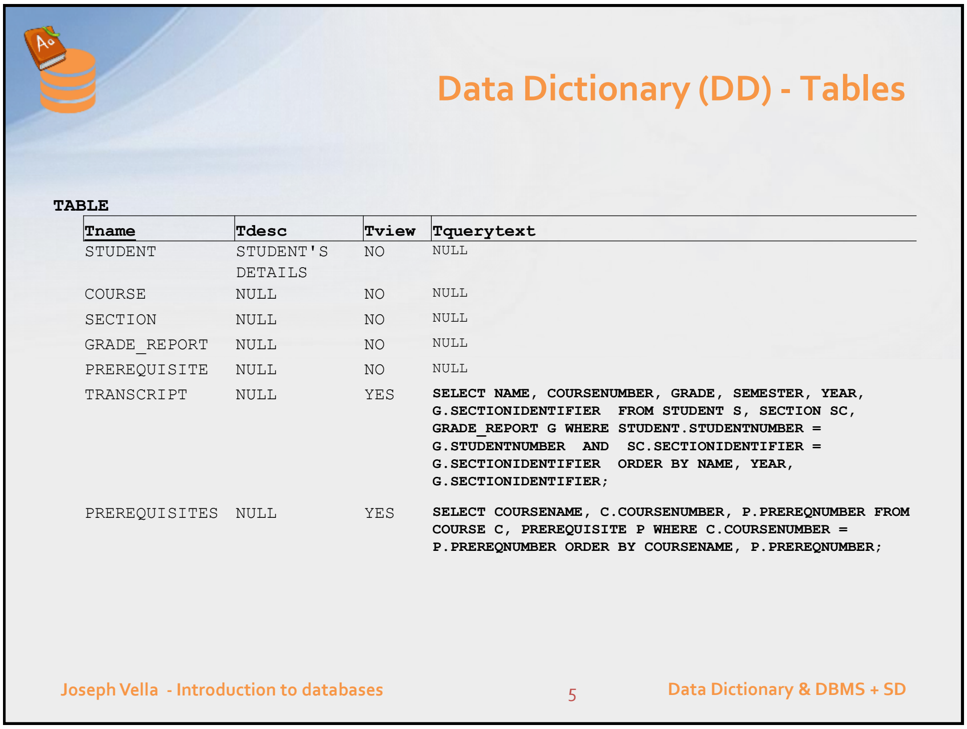
STUDENT	Name	StudentNumber	Class	Major
	Smith	17	1	CS
	Brown	8	2	CS

COURSE	CourseName	CourseNumber	CreditHours	Department
	Intro to Computer Science	CS1310	4	CS
	Data Structures	CS3320	4	CS
	Discrete Mathematics	MATH2410	3	MATH
	Database	CS3380	3	CS

SECTION	SectionIdentifier	CourseNumber	Semester	Year	Instructor
	85	MATH2410	Fall	98	King
	92	CS1310	Fall	98	Anderson
	102	CS3320	Spring	99	Knuth
	112	MATH2410	Fall	99	Chang
	119	CS1310	Fall	99	Anderson
	135	CS3380	Fall	99	Stone

GRADE_REPORT	StudentNumber	SectionIdentifier	Grade
	17	112	B
	17	119	C
	8	85	A
	8	92	A
	8	102	B
	8	135	A

PREREQUISITE	CourseNumber	PrerequisiteNumber
	CS3380	CS3320
	CS3380	MATH2410
	CS3320	CS1310



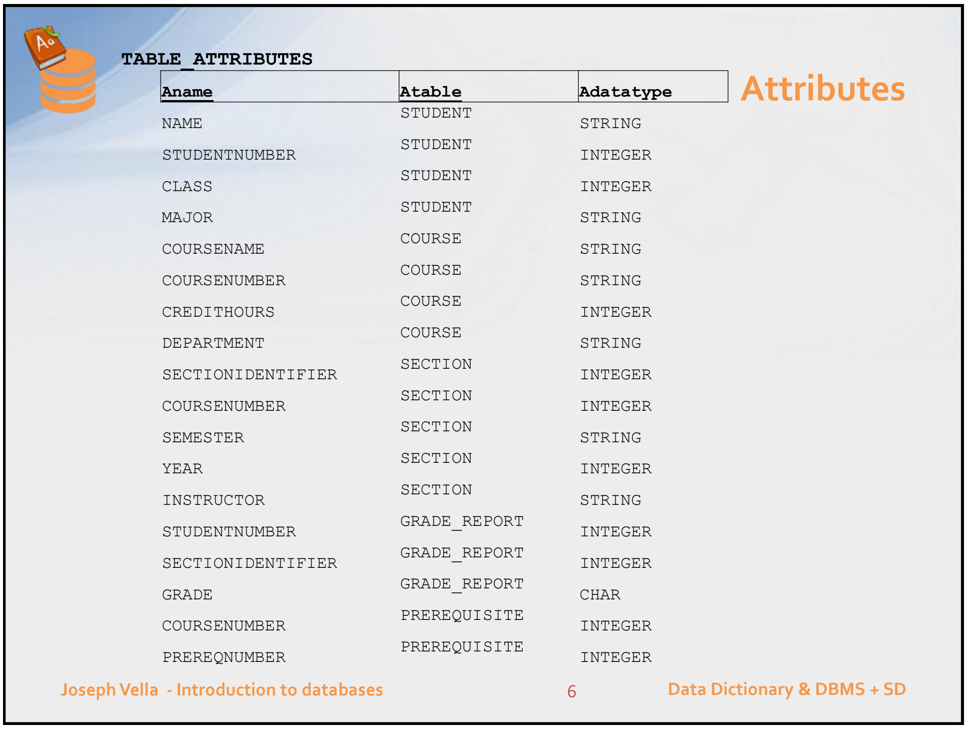
Data Dictionary (DD) - Tables

TABLE

Tname	Tdesc	Tview	Tquerytext
STUDENT	STUDENT'S DETAILS	NO	NULL
COURSE	NULL	NO	NULL
SECTION	NULL	NO	NULL
GRADE_REPORT	NULL	NO	NULL
PREREQUISITE	NULL	NO	NULL
TRANSCRIPT	NULL	YES	SELECT NAME, COURSENUMBER, GRADE, SEMESTER, YEAR, G.SECTIONIDENTIFIER FROM STUDENT S, SECTION SC, GRADE_REPORT G WHERE STUDENT.STUDENTNUMBER = G.STUDENTNUMBER AND SC.SECTIONIDENTIFIER = G.SECTIONIDENTIFIER ORDER BY NAME, YEAR, G.SECTIONIDENTIFIER;
PREREQUISITES	NULL	YES	SELECT COURSENAME, C.COURSENUMBER, P.PREREQNUMBER FROM COURSE C, PREREQUISITE P WHERE C.COURSENUMBER = P.PREREQNUMBER ORDER BY COURSENAME, P.PREREQNUMBER;

Joseph Vella - Introduction to databases 5 Data Dictionary & DBMS + SD

5



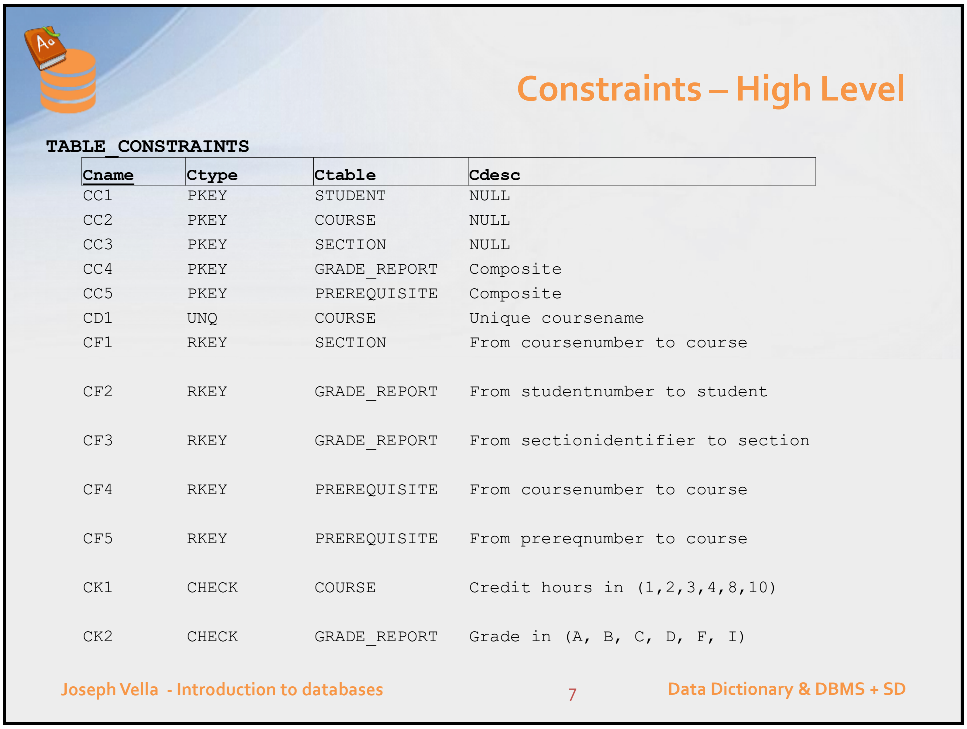
Attributes

TABLE_ATTRIBUTES

Aname	Atable	Adatatype
NAME	STUDENT	STRING
STUDENTNUMBER	STUDENT	INTEGER
CLASS	STUDENT	INTEGER
MAJOR	STUDENT	STRING
COURSENAME	COURSE	STRING
COURSENUMBER	COURSE	STRING
CREDITHOURS	COURSE	INTEGER
DEPARTMENT	COURSE	STRING
SECTIONIDENTIFIER	SECTION	INTEGER
COURSENUMBER	SECTION	INTEGER
SEMESTER	SECTION	STRING
YEAR	SECTION	INTEGER
INSTRUCTOR	SECTION	STRING
STUDENTNUMBER	GRADE_REPORT	INTEGER
SECTIONIDENTIFIER	GRADE_REPORT	INTEGER
GRADE	GRADE_REPORT	CHAR
COURSENUMBER	PREREQUISITE	INTEGER
PREREQNUMBER	PREREQUISITE	INTEGER

Joseph Vella - Introduction to databases 6 Data Dictionary & DBMS + SD

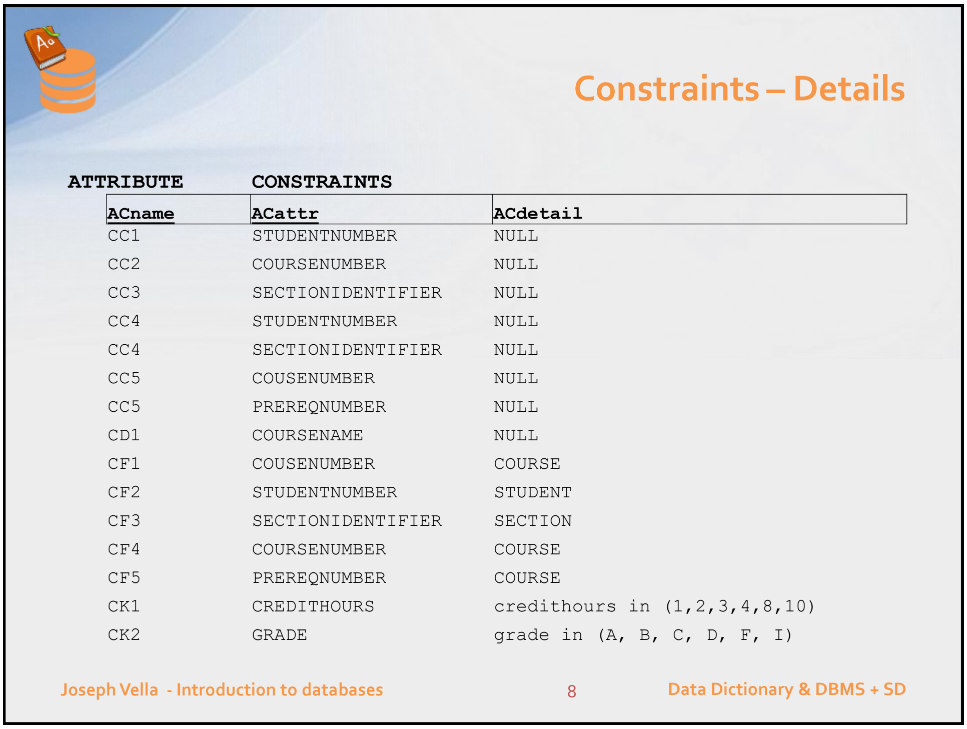
6



Constraints – High Level

TABLE_CONSTRAINTS

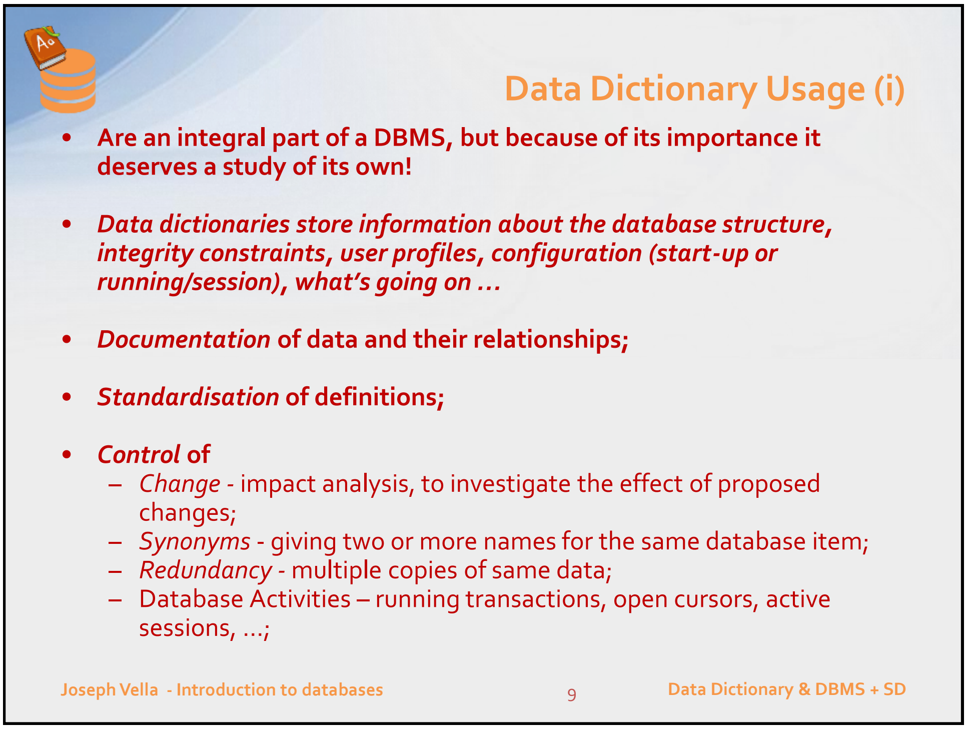
Cname	Ctype	Ctable	Cdesc
CC1	PKEY	STUDENT	NULL
CC2	PKEY	COURSE	NULL
CC3	PKEY	SECTION	NULL
CC4	PKEY	GRADE_REPORT	Composite
CC5	PKEY	PREREQUISITE	Composite
CD1	UNQ	COURSE	Unique coursename
CF1	RKEY	SECTION	From coursenumber to course
CF2	RKEY	GRADE_REPORT	From studentnumber to student
CF3	RKEY	GRADE_REPORT	From sectionidentifier to section
CF4	RKEY	PREREQUISITE	From coursenumber to course
CF5	RKEY	PREREQUISITE	From prereqnumber to course
CK1	CHECK	COURSE	Credit hours in (1,2,3,4,8,10)
CK2	CHECK	GRADE_REPORT	Grade in (A, B, C, D, F, I)



Constraints – Details

ATTRIBUTE CONSTRAINTS

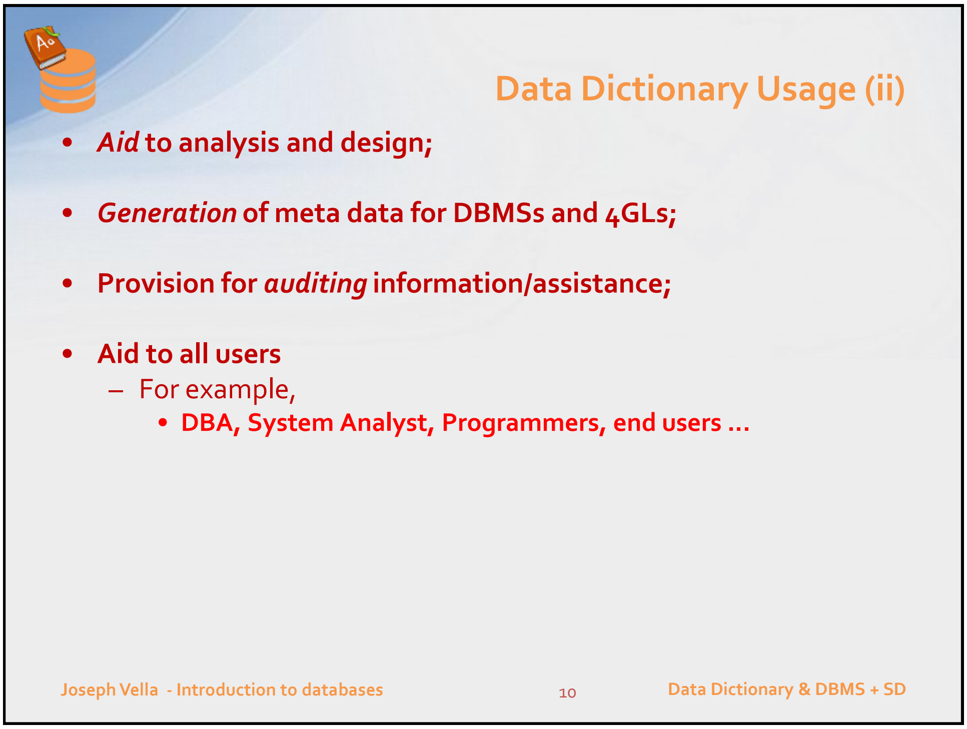
ACname	ACattr	ACdetail
CC1	STUDENTNUMBER	NULL
CC2	COURSENUMBER	NULL
CC3	SECTIONIDENTIFIER	NULL
CC4	STUDENTNUMBER	NULL
CC4	SECTIONIDENTIFIER	NULL
CC5	COUSENUMBER	NULL
CC5	PREREQNUMBER	NULL
CD1	COURSENAME	NULL
CF1	COUSENUMBER	COURSE
CF2	STUDENTNUMBER	STUDENT
CF3	SECTIONIDENTIFIER	SECTION
CF4	COURSENUMBER	COURSE
CF5	PREREQNUMBER	COURSE
CK1	CREDITHOURS	credithours in (1,2,3,4,8,10)
CK2	GRADE	grade in (A, B, C, D, F, I)



Data Dictionary Usage (i)

- Are an integral part of a DBMS, but because of its importance it deserves a study of its own!
- *Data dictionaries store information about the database structure, integrity constraints, user profiles, configuration (start-up or running/session), what's going on …*
- *Documentation* of data and their relationships;
- *Standardisation* of definitions;
- *Control* of
  - *Change* - impact analysis, to investigate the effect of proposed changes;
  - *Synonyms* - giving two or more names for the same database item;
  - *Redundancy* - multiple copies of same data;
  - Database Activities – running transactions, open cursors, active sessions, …;

9

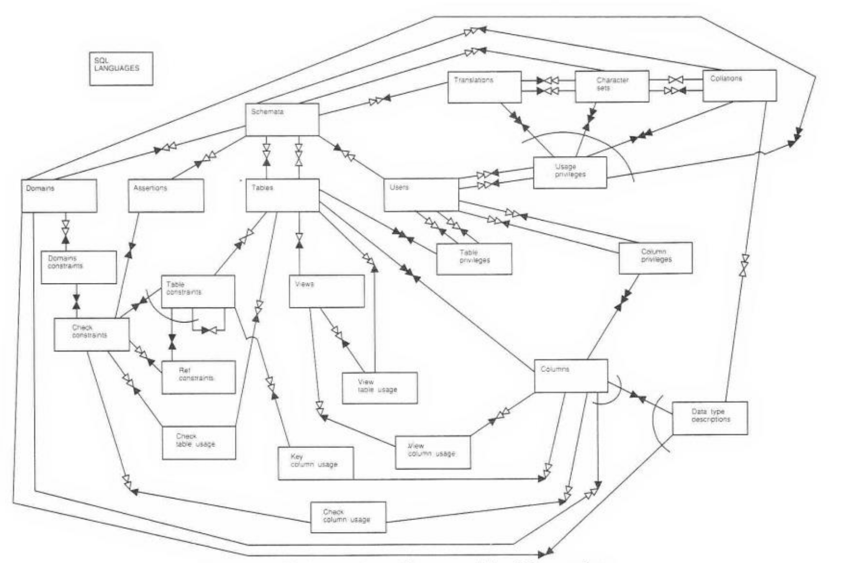


Data Dictionary Usage (ii)

- *Aid* to analysis and design;
- *Generation* of meta data for DBMSs and 4GLs;
- Provision for *auditing* information/assistance;
- Aid to all users
  - For example,
    - DBA, System Analyst, Programmers, end users …

10

ERM of data dictionary from Cotton's Book on SQLII

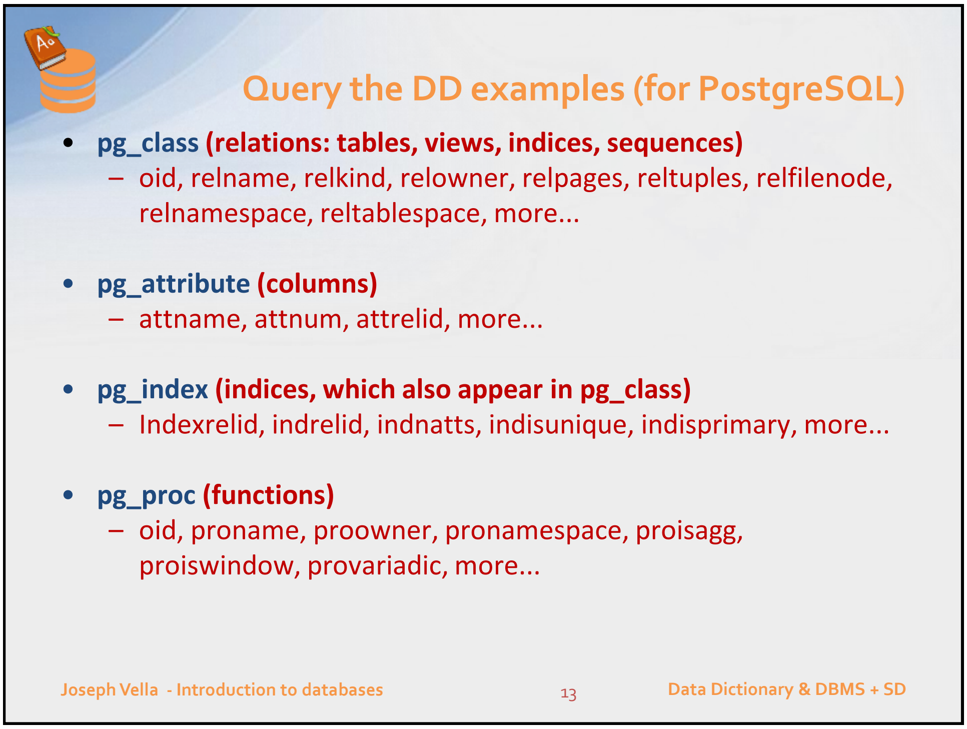


Data structure diagram of the SQL catalog

PostgreSQL & Data Dictionary

- System tables (catalogs), views, and functions exist within a special schema called pg_catalog.

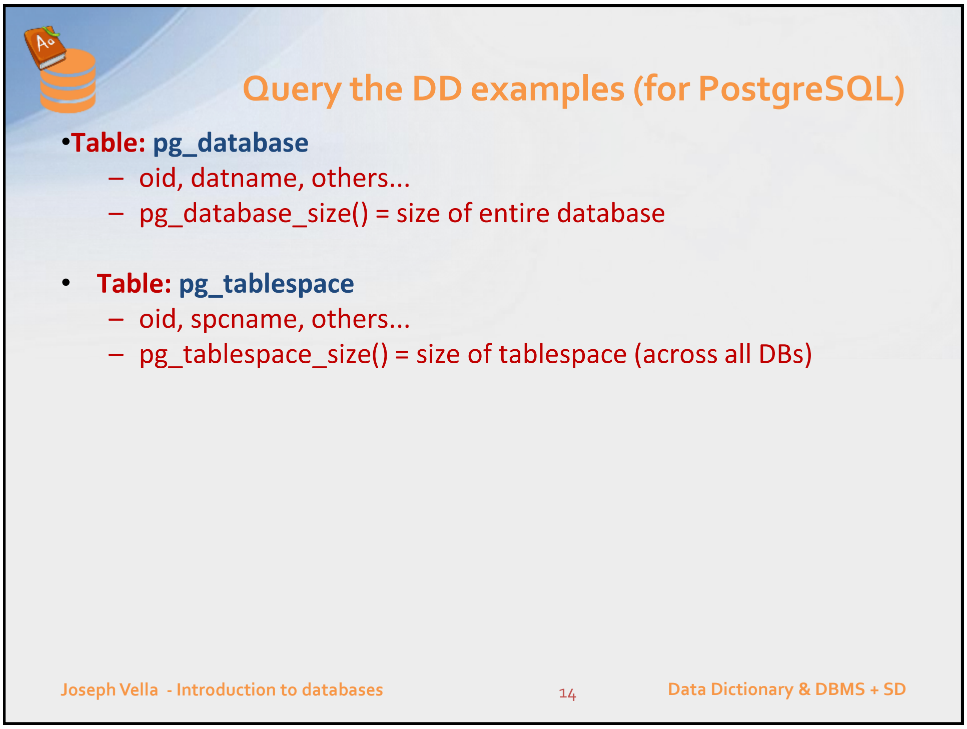
- Some system objects are there for your benefit (e.g. pg_database_size).
    - System tables (catalogs) are somewhere in the middle.



Query the DD examples (for PostgreSQL)

- **pg_class (relations: tables, views, indices, sequences)**
  - oid, relname, relkind, relowner, relpages, reltuples, relfilenode, relnamespace, reltablespace, more...
- **pg_attribute (columns)**
  - attname, attnum, attrelid, more...
- **pg_index (indices, which also appear in pg_class)**
  - Indexrelid, indrelid, indnatts, indisunique, indisprimary, more...
- **pg_proc (functions)**
  - oid, proname, proowner, pronamespace, proisagg, proiswindow, provariadic, more...

13

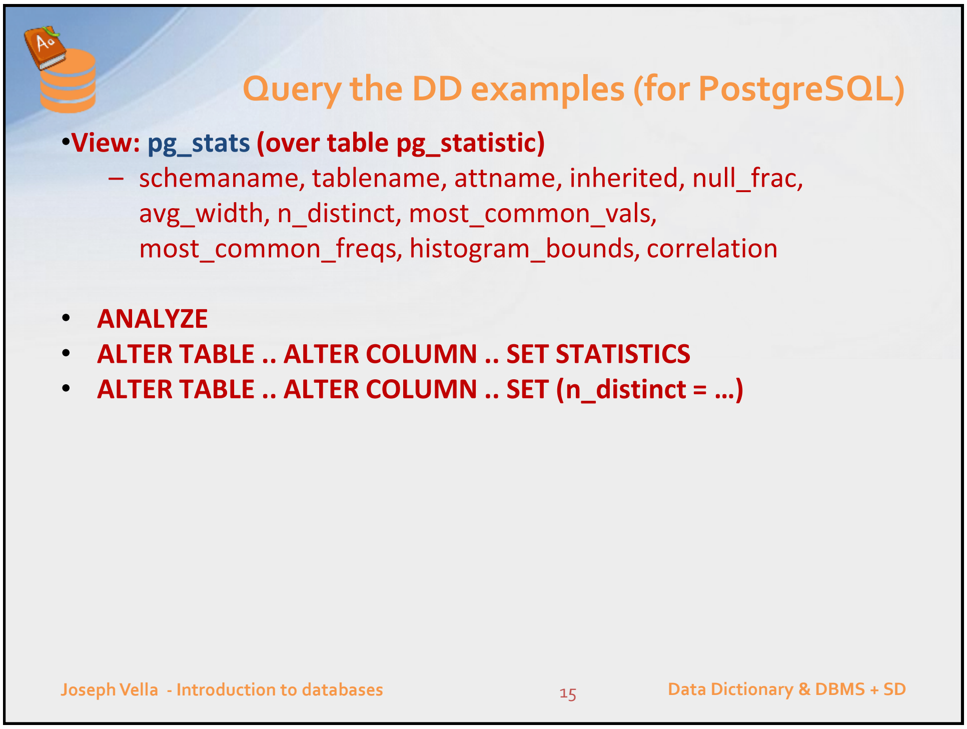


Query the DD examples (for PostgreSQL)

•**Table: pg_database**
  - oid, datname, others...
  - pg_database_size() = size of entire database

- **Table: pg_tablespace**
  - oid, spcname, others...
  - pg_tablespace_size() = size of tablespace (across all DBs)

14

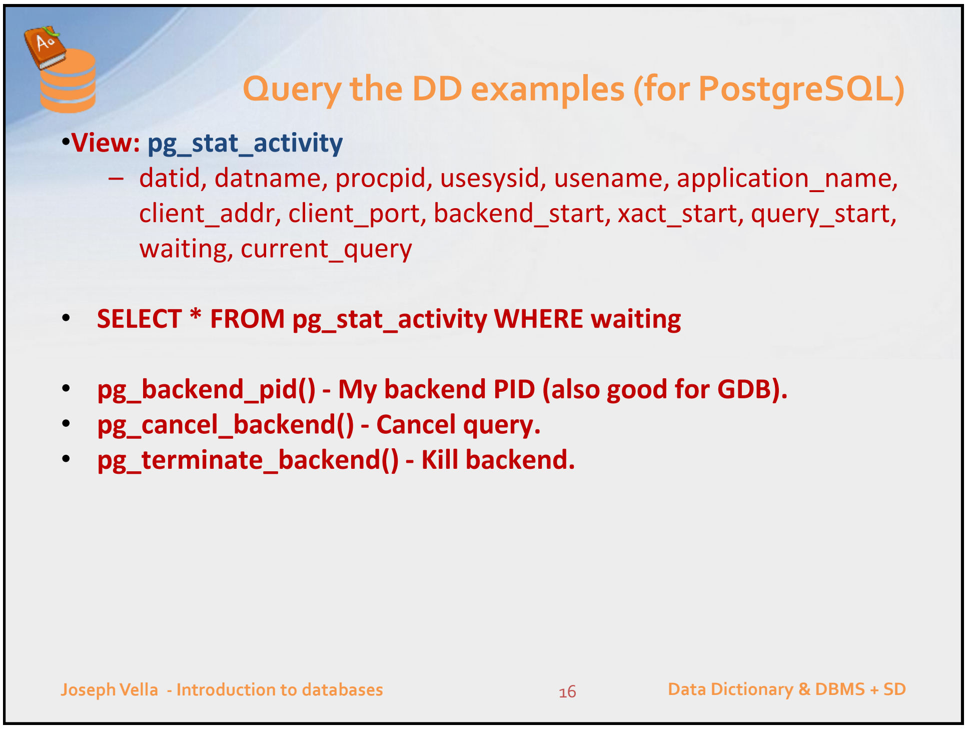


Query the DD examples (for PostgreSQL)

- **View: pg_stats (over table pg_statistic)**
  - schemaname, tablename, attname, inherited, null_frac, avg_width, n_distinct, most_common_vals, most_common_freqs, histogram_bounds, correlation

- **ANALYZE**
- **ALTER TABLE .. ALTER COLUMN .. SET STATISTICS**
- **ALTER TABLE .. ALTER COLUMN .. SET (n_distinct = …)**

15



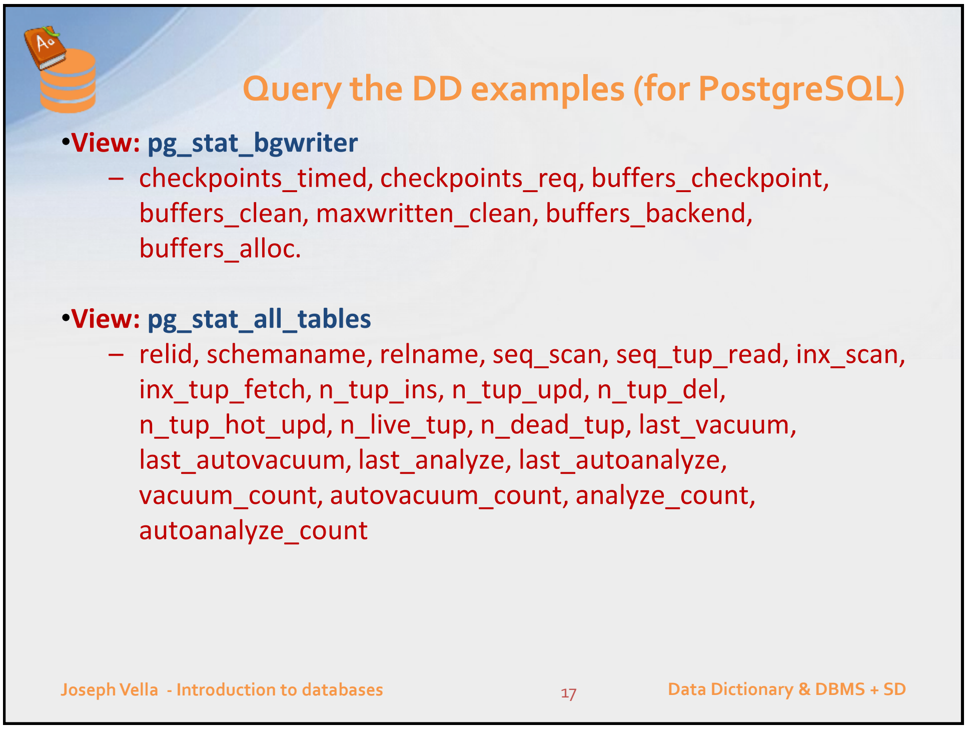
Query the DD examples (for PostgreSQL)

- **View: pg_stat_activity**
  - datid, datname, procpid, usesysid, usename, application_name, client_addr, client_port, backend_start, xact_start, query_start, waiting, current_query

- **SELECT * FROM pg_stat_activity WHERE waiting**

- **pg_backend_pid() - My backend PID (also good for GDB).**
- **pg_cancel_backend() - Cancel query.**
- **pg_terminate_backend() - Kill backend.**

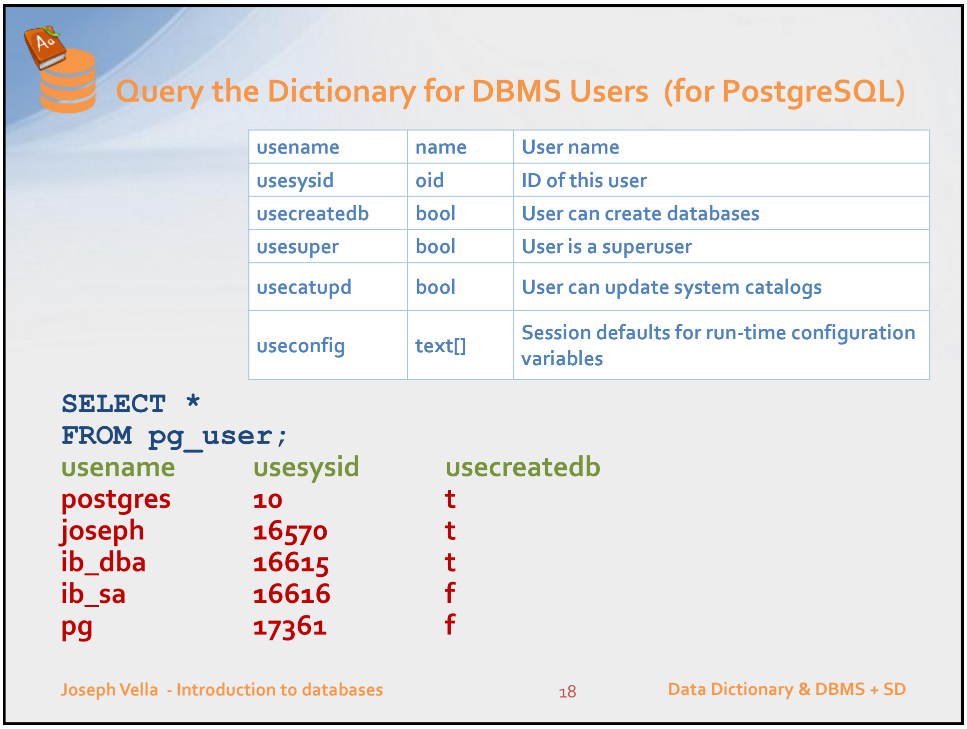
16



Query the DD examples (for PostgreSQL)

- **View: pg_stat_bgwriter**
  - checkpoints_timed, checkpoints_req, buffers_checkpoint, buffers_clean, maxwritten_clean, buffers_backend, buffers_alloc.
- **View: pg_stat_all_tables**
  - relid, schemaname, relname, seq_scan, seq_tup_read, inx_scan, inx_tup_fetch, n_tup_ins, n_tup_upd, n_tup_del, n_tup_hot_upd, n_live_tup, n_dead_tup, last_vacuum, last_autovacuum, last_analyze, last_autoanalyze, vacuum_count, autovacuum_count, analyze_count, autoanalyze_count

17



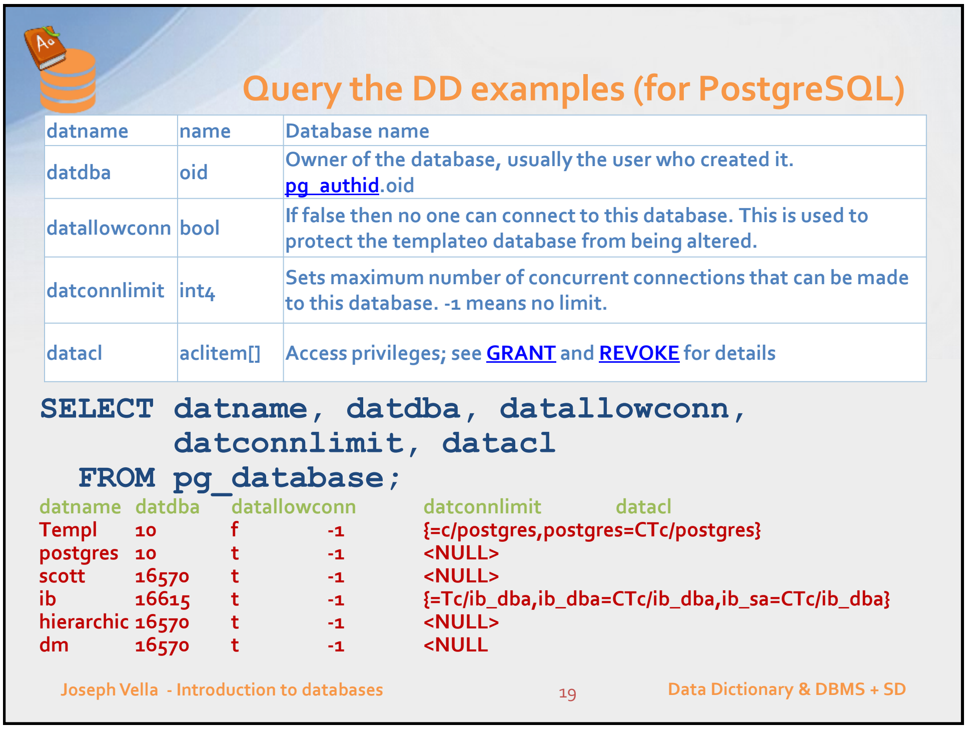
Query the Dictionary for DBMS Users (for PostgreSQL)

usename	name	User name
usesysid	oid	ID of this user
usecreatedb	bool	User can create databases
usesuper	bool	User is a superuser
usecatupd	bool	User can update system catalogs
useconfig	text[]	Session defaults for run-time configuration variables

```
SELECT *
FROM pg_user;
```

```
usename     usesysid    usecreatedb
postgres    10          t
joseph      16570       t
ib_dba      16615       t
ib_sa       16616       f
pg          17361       f
```

18

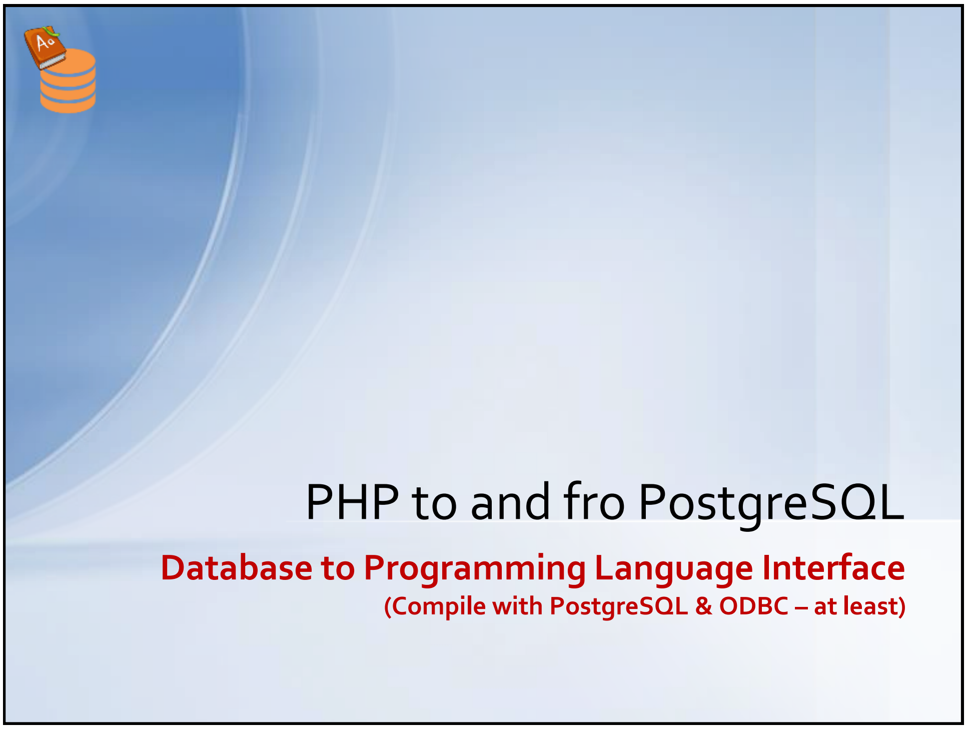


Query the DD examples (for PostgreSQL)

datname	name	Database name
datdba	oid	Owner of the database, usually the user who created it. pg_authid.oid
datallowconn	bool	If false then no one can connect to this database. This is used to protect the templateo database from being altered.
datconnlimit	int4	Sets maximum number of concurrent connections that can be made to this database. -1 means no limit.
datacl	aclitem[]	Access privileges; see GRANT and REVOKE for details

```
SELECT datname, datdba, datallowconn,
       datconnlimit, datacl
  FROM pg_database;
```

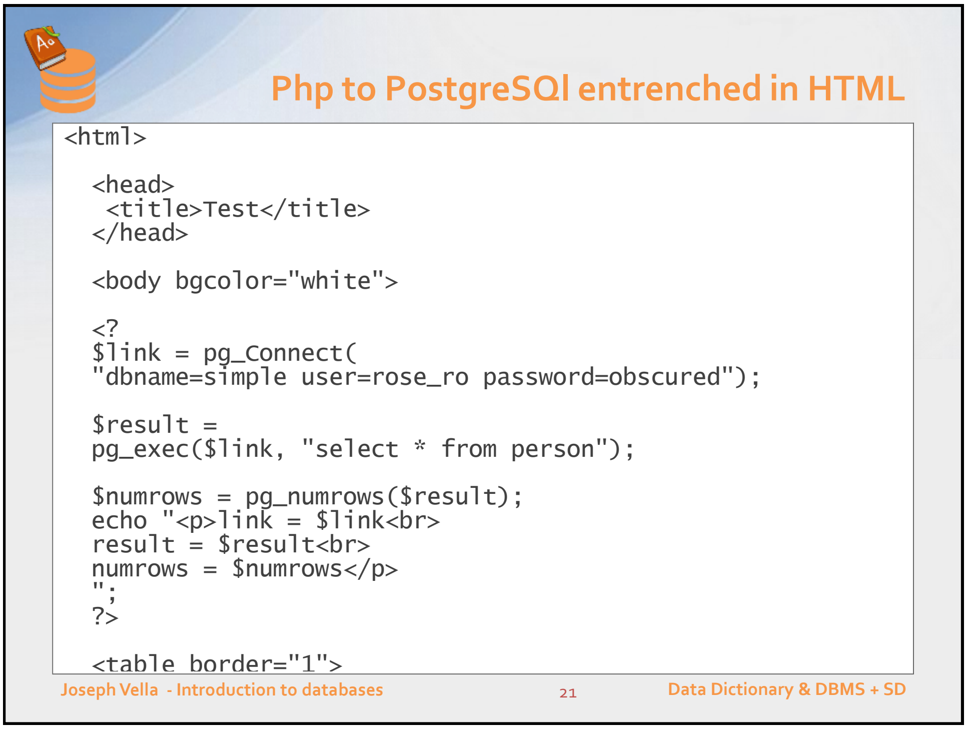
```
datname   datdba   datallowconn   datconnlimit   datacl
Templ     10       f              -1             {=c/postgres,postgres=CTc/postgres}
postgres  10       t              -1             <NULL>
scott     16570    t              -1             <NULL>
ib        16615    t              -1             {=Tc/ib_dba,ib_dba=CTc/ib_dba,ib_sa=CTc/ib_dba}
hierarchic 16570   t              -1             <NULL>
dm        16570    t              -1             <NULL
```



PHP to and fro PostgreSQL

Database to Programming Language Interface

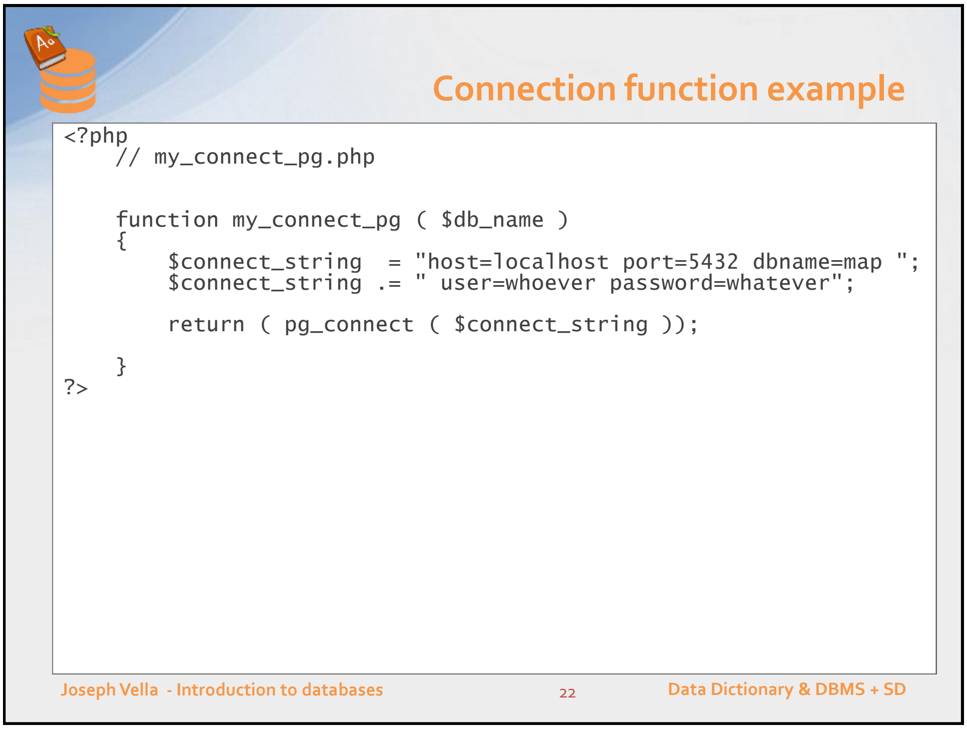
(Compile with PostgreSQL & ODBC – at least)



Php to PostgreSQl entrenched in HTML

```
<html>

  <head>
   <title>Test</title>
  </head>

  <body bgcolor="white">

  <?
  $link = pg_Connect(
  "dbname=simple user=rose_ro password=obscured");

  $result =
  pg_exec($link, "select * from person");

  $numrows = pg_numrows($result);
  echo "<p>link = $link<br>
  result = $result<br>
  numrows = $numrows</p>
  ";
  ?>

  <table border="1">
```

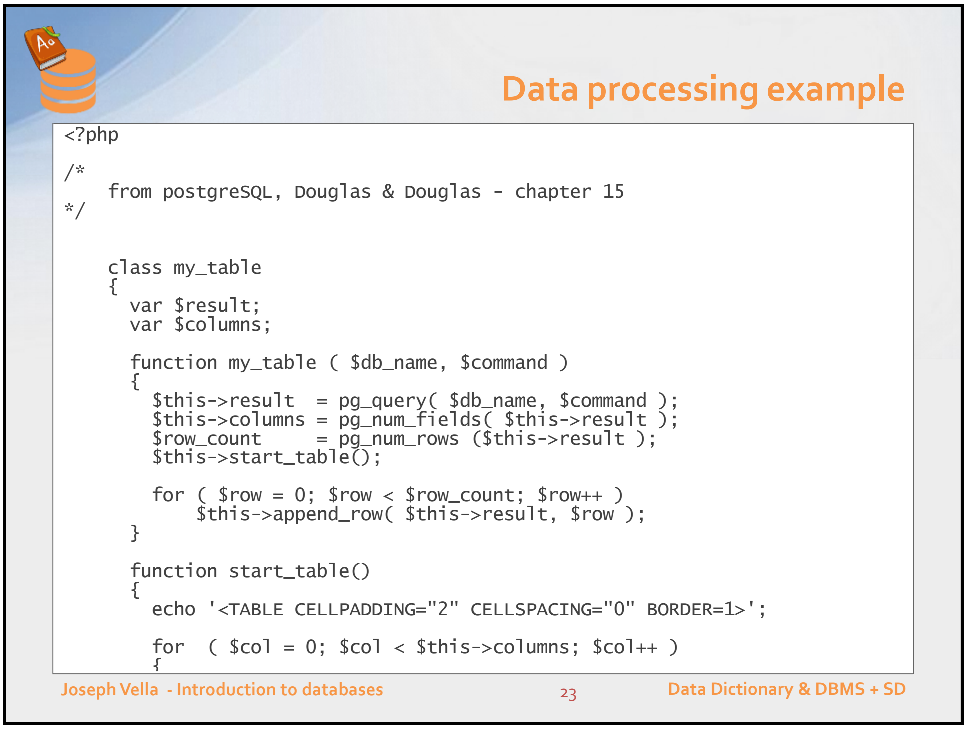
21



Connection function example

```
<?php
    // my_connect_pg.php


    function my_connect_pg ( $db_name )
    {
        $connect_string  = "host=localhost port=5432 dbname=map ";
        $connect_string .= " user=whoever password=whatever";

        return ( pg_connect ( $connect_string ));

    }
?>
```

22

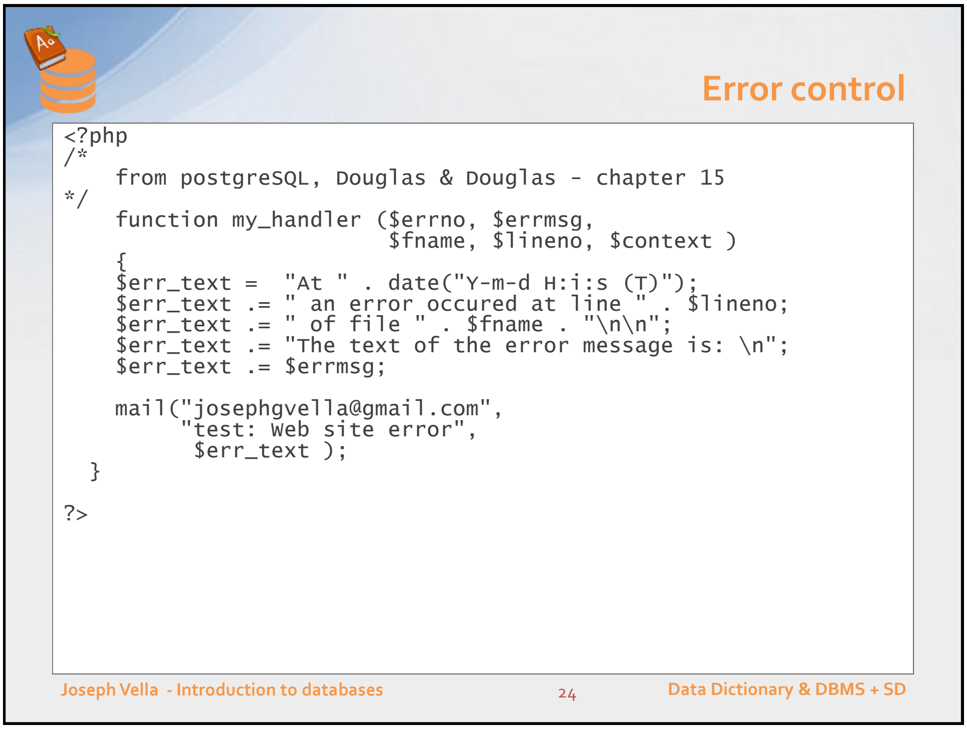


Data processing example

```
<?php
/*
    from postgreSQL, Douglas & Douglas - chapter 15
*/


    class my_table
    {
      var $result;
      var $columns;

      function my_table ( $db_name, $command )
      {
        $this->result  = pg_query( $db_name, $command );
        $this->columns = pg_num_fields( $this->result );
        $row_count     = pg_num_rows ($this->result );
        $this->start_table();

        for ( $row = 0; $row < $row_count; $row++ )
            $this->append_row( $this->result, $row );
      }

      function start_table()
      {
        echo '<TABLE CELLPADDING="2" CELLSPACING="0" BORDER=1>';

        for  ( $col = 0; $col < $this->columns; $col++ )
        {
```

Joseph Vella - Introduction to databases 23 Data Dictionary & DBMS + SD

23



Error control

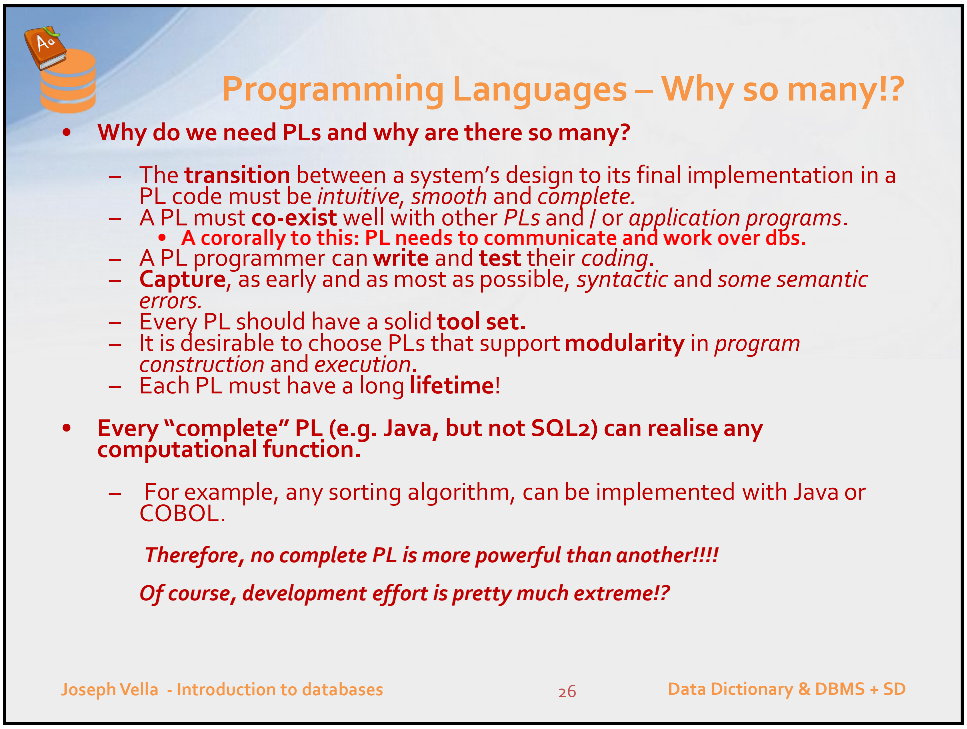
```
<?php
/*
    from postgreSQL, Douglas & Douglas - chapter 15
*/
    function my_handler ($errno, $errmsg,
                         $fname, $lineno, $context )
    {
    $err_text =  "At " . date("Y-m-d H:i:s (T)");
    $err_text .= " an error occured at line " . $lineno;
    $err_text .= " of file " . $fname . "\n\n";
    $err_text .= "The text of the error message is: \n";
    $err_text .= $errmsg;

    mail("josephgvella@gmail.com",
         "test: Web site error",
          $err_text );
  }

?>
```

Joseph Vella - Introduction to databases 24 Data Dictionary & DBMS + SD

24

Beautiful Code

25



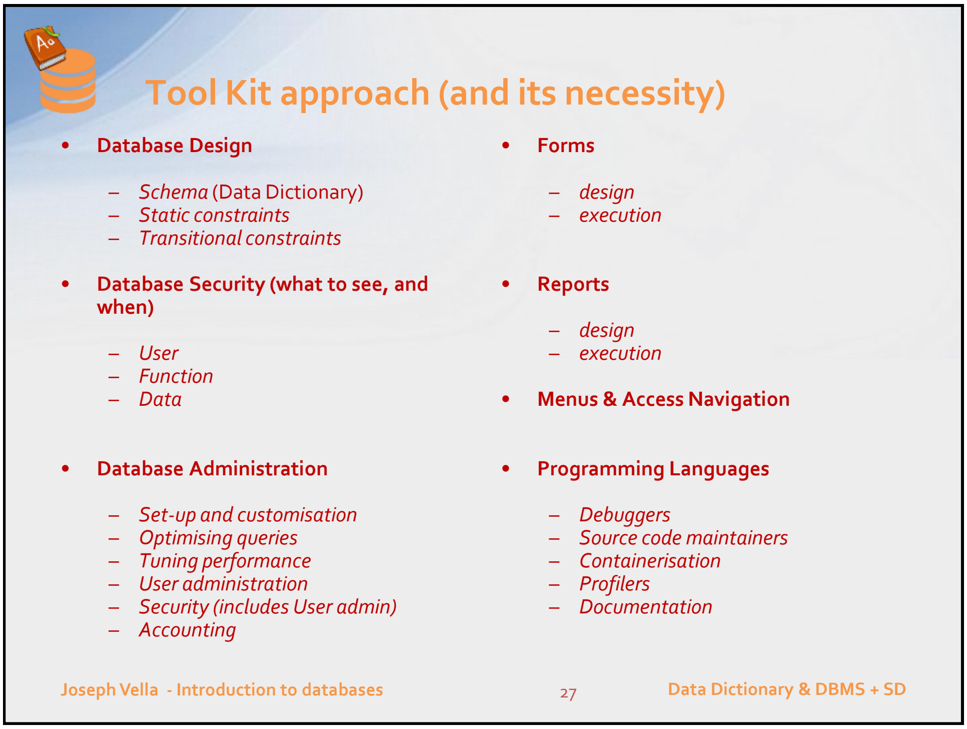
Programming Languages – Why so many!?

- **Why do we need PLs and why are there so many?**
  - The **transition** between a system's design to its final implementation in a PL code must be *intuitive*, *smooth* and *complete*.
  - A PL must **co-exist** well with other *PLs* and / or *application programs*.
    - **A cororally to this: PL needs to communicate and work over dbs.**
  - A PL programmer can **write** and **test** their *coding*.
  - **Capture**, as early and as most as possible, *syntactic* and *some semantic errors*.
  - Every PL should have a solid **tool set.**
  - It is desirable to choose PLs that support **modularity** in *program construction* and *execution*.
  - Each PL must have a long **lifetime**!
- **Every "complete" PL (e.g. Java, but not SQL2) can realise any computational function.**
  - For example, any sorting algorithm, can be implemented with Java or COBOL.

    Therefore, no complete PL is more powerful than another!!!!

    Of course, development effort is pretty much extreme!?

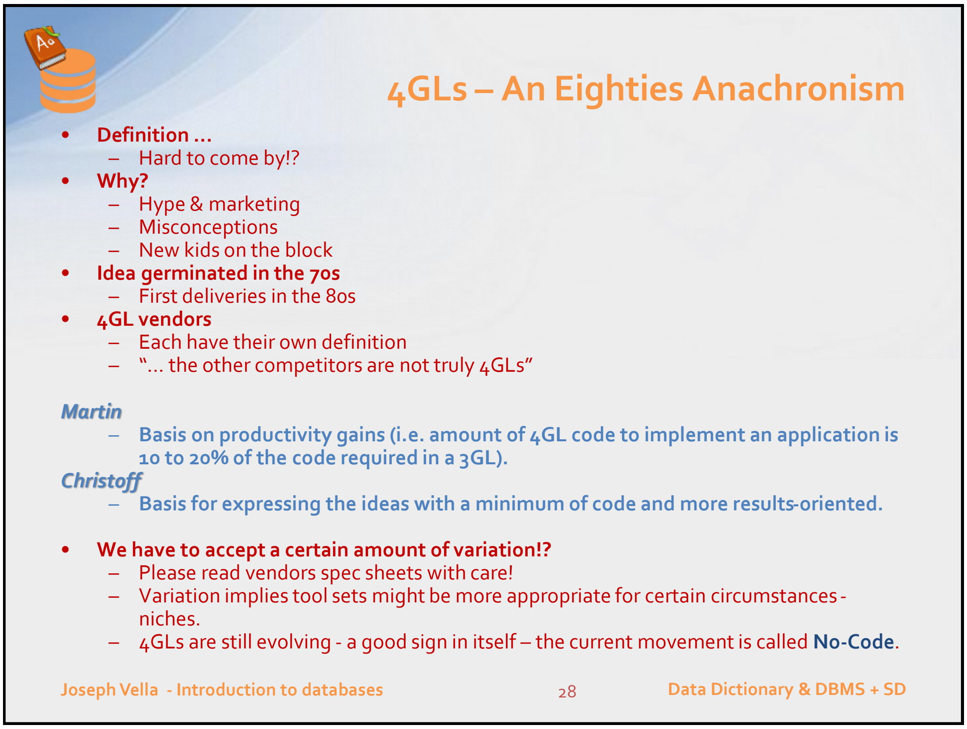
26



Tool Kit approach (and its necessity)

- **Database Design**
  - *Schema* (Data Dictionary)
  - *Static constraints*
  - *Transitional constraints*
- **Database Security (what to see, and when)**
  - *User*
  - *Function*
  - *Data*
- **Database Administration**
  - *Set-up and customisation*
  - *Optimising queries*
  - *Tuning performance*
  - *User administration*
  - *Security (includes User admin)*
  - *Accounting*
- **Forms**
  - *design*
  - *execution*
- **Reports**
  - *design*
  - *execution*
- **Menus & Access Navigation**
- **Programming Languages**
  - *Debuggers*
  - *Source code maintainers*
  - *Containerisation*
  - *Profilers*
  - *Documentation*

27



4GLs – An Eighties Anachronism

- **Definition …**
  - Hard to come by!?
- **Why?**
  - Hype & marketing
  - Misconceptions
  - New kids on the block
- **Idea germinated in the 70s**
  - First deliveries in the 80s
- **4GL vendors**
  - Each have their own definition
  - "… the other competitors are not truly 4GLs"

Martin
  - Basis on productivity gains (i.e. amount of 4GL code to implement an application is 10 to 20% of the code required in a 3GL).

Christoff
  - Basis for expressing the ideas with a minimum of code and more results-oriented.

- **We have to accept a certain amount of variation!?**
  - Please read vendors spec sheets with care!
  - Variation implies tool sets might be more appropriate for certain circumstances - niches.
  - 4GLs are still evolving - a good sign in itself – the current movement is called **No-Code**.

28

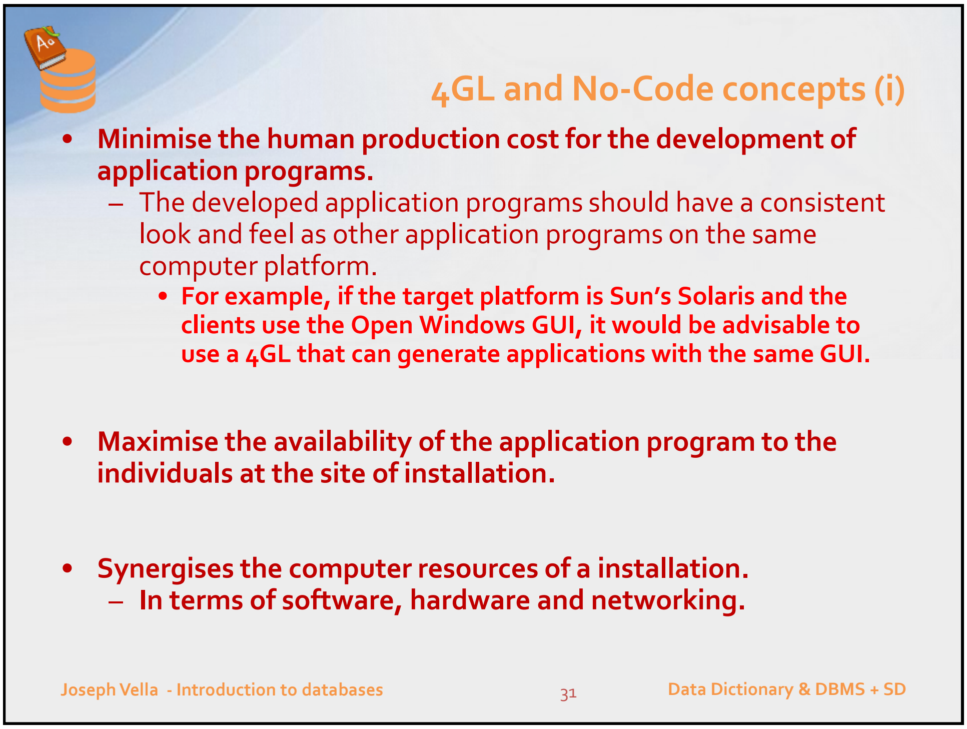
FYI

What is no-code development?
Build a wide range of apps for internal business users and external customers.
Integration with third-party tools and services
Preconfigured back-end connectivity
Build mobile apps, websites and other systems
Intuitive drag-and-drop operations
Reduced costs and increased flexibility
Feature-rich visual development environment

https://www.techtarget.com/searchsoftwarequality/definition/no-code

4GL and No-Code Characteristics

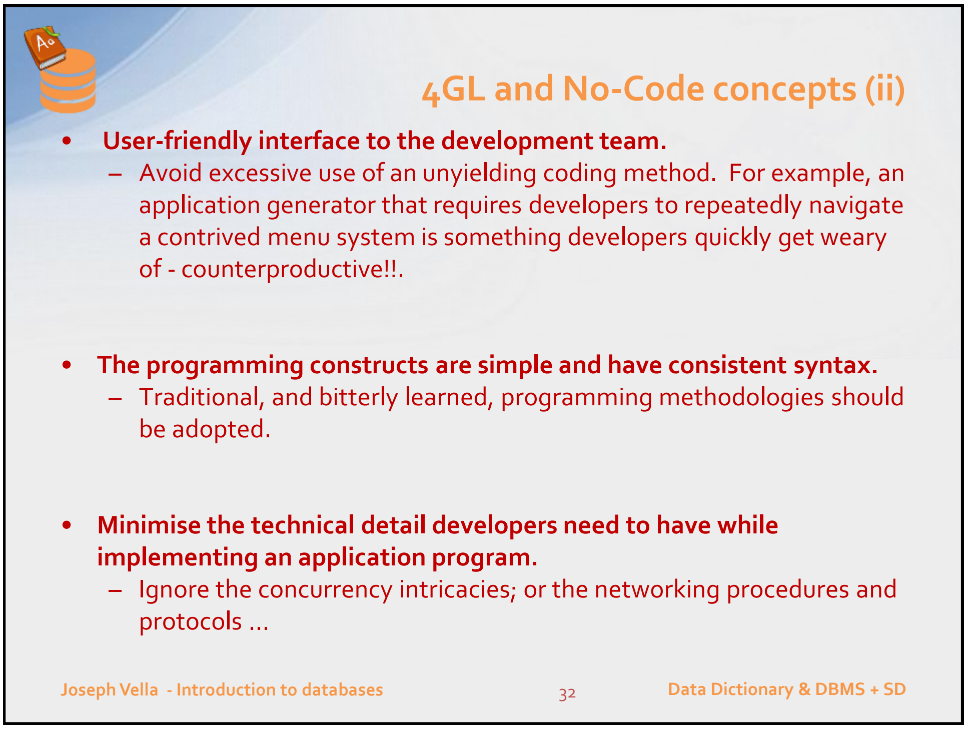
- **High Level Language**
  - High level PLs have constructs that are like the English language constructs.
    4GLs are definitely high level.
  - Low level PLs have constructs that directly access the computer's hardware devices or required APIs.
- **Easy to 'Write'**
  - Straight forward and consistent syntax/activities.
- **Easy to Maintain**
  - Adequate support for program "reading" and documentation.
- **Cohesive**
  - Fits seamlessly with other PLs and / or 4GLs.
- **Prototyping**
  - Developer and end user jointly develop an embryonic application program facet -- this product is not sufficient but can be easily changed until it satisfies the users' expectations.
- **Database Culture**
  - Dealing with a magnitude of ever present structured and unstructured data.
  - Managing a high number of concurrent users each with his own priorities.



4GL and No-Code concepts (i)

- **Minimise the human production cost for the development of application programs.**
  - The developed application programs should have a consistent look and feel as other application programs on the same computer platform.
    - **For example, if the target platform is Sun's Solaris and the clients use the Open Windows GUI, it would be advisable to use a 4GL that can generate applications with the same GUI.**
- **Maximise the availability of the application program to the individuals at the site of installation.**
- **Synergises the computer resources of a installation.**
  - **In terms of software, hardware and networking.**

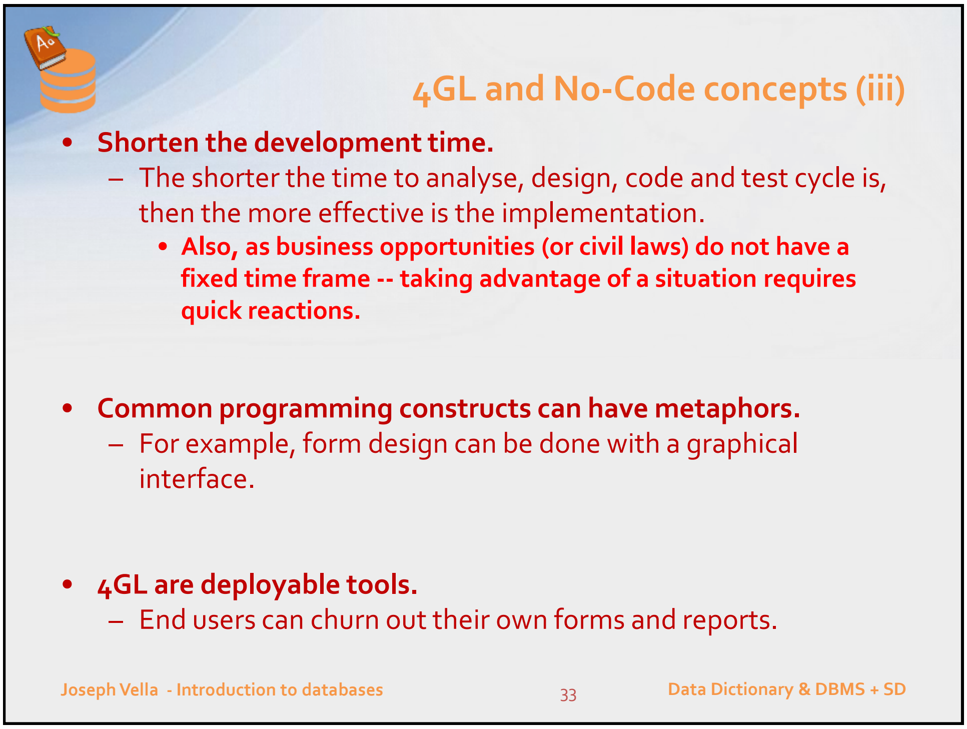
31



4GL and No-Code concepts (ii)

- **User-friendly interface to the development team.**
  - Avoid excessive use of an unyielding coding method. For example, an application generator that requires developers to repeatedly navigate a contrived menu system is something developers quickly get weary of - counterproductive!!.
- **The programming constructs are simple and have consistent syntax.**
  - Traditional, and bitterly learned, programming methodologies should be adopted.
- **Minimise the technical detail developers need to have while implementing an application program.**
  - Ignore the concurrency intricacies; or the networking procedures and protocols …

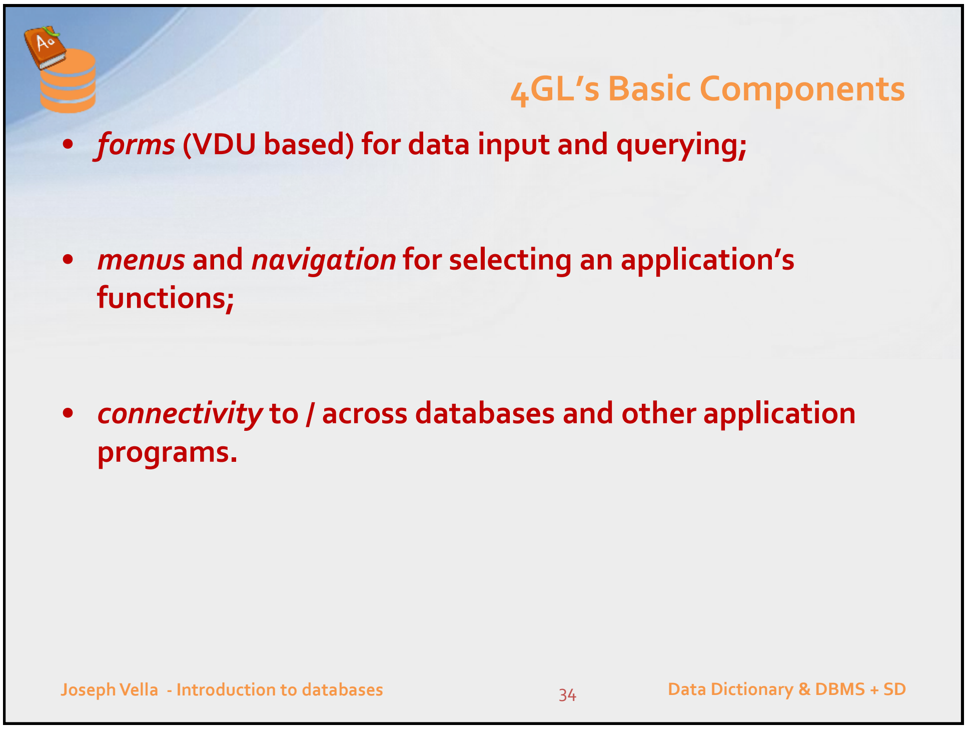
32



4GL and No-Code concepts (iii)

- **Shorten the development time.**
  - The shorter the time to analyse, design, code and test cycle is, then the more effective is the implementation.
    - **Also, as business opportunities (or civil laws) do not have a fixed time frame -- taking advantage of a situation requires quick reactions.**
- **Common programming constructs can have metaphors.**
  - For example, form design can be done with a graphical interface.
- **4GL are deployable tools.**
  - End users can churn out their own forms and reports.

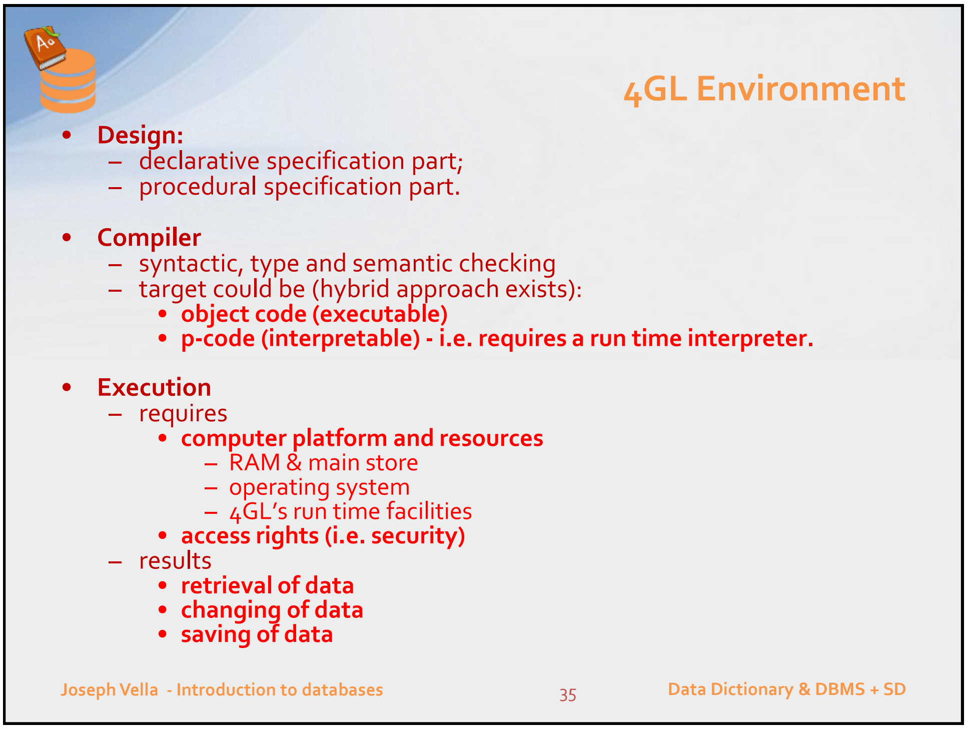
33



4GL's Basic Components

- ***forms* (VDU based) for data input and querying;**
- ***menus* and *navigation* for selecting an application's functions;**
- ***connectivity* to / across databases and other application programs.**

34



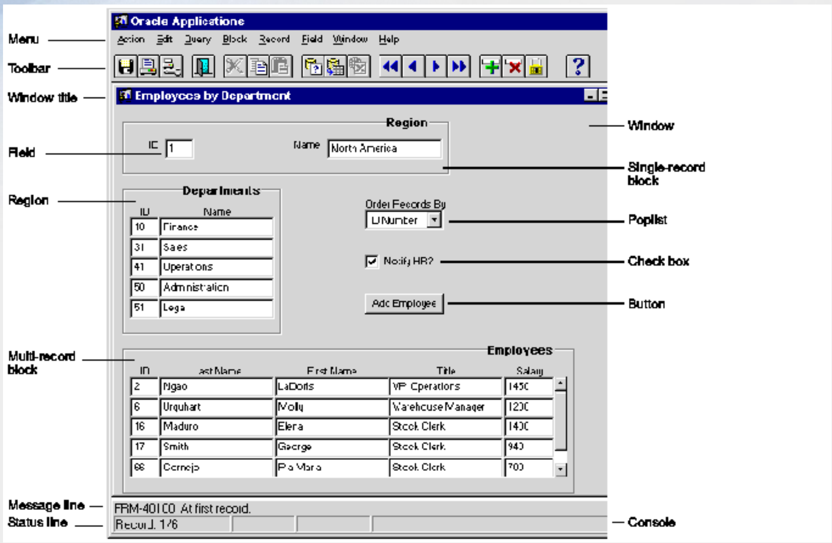
4GL Environment

- **Design:**
  - declarative specification part;
  - procedural specification part.
- **Compiler**
  - syntactic, type and semantic checking
  - target could be (hybrid approach exists):
    - **object code (executable)**
    - **p-code (interpretable) - i.e. requires a run time interpreter.**
- **Execution**
  - requires
    - **computer platform and resources**
      - RAM & main store
      - operating system
      - 4GL's run time facilities
    - **access rights (i.e. security)**
  - results
    - **retrieval of data**
    - **changing of data**
    - **saving of data**

35

Forms

36



Region

IC 1 Name North America

Departments

ID	Name
10	Finance
31	Saes
41	Uperat ons
50	Adm nist alicn
51	Lega

Order Records By: LNumber

☑ Notify HR?

Add Employee

Employees

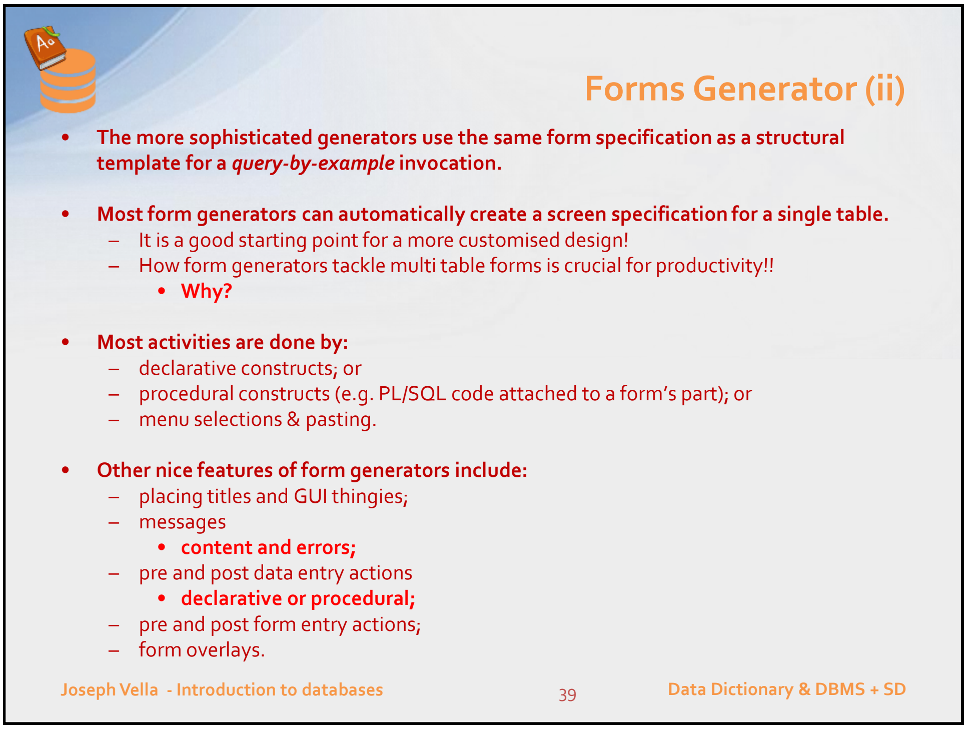
ID	ast Name	First Name	Title	Salary
2	Ngao	LaDoris	VP Cperatiors	145C
6	Urquhart	Molly	Warehcuse Manager	120C
16	Maduro	Elena	Stock Clerk	140C
17	Smith	George	Stock Clerk	940
66	Cornejo	Pia Mara	Stock Clerk	700

FRM-40100 At first record.

Record: 1/6

Forms Generators (i)

- 4GLs offer *automated utilities to "paint"* the screens interactively.
- Most form generators do more than automate painting!
- Approaches found across a number of 4GLs include:
  - *A screen mapper*
    - Draws the layout of the screen and clearly delineates between fixed and variable data. Also, it is important to sequence the access patterns.
  - *A map editor*
    - Draws the screen layout in the same way the screen mapper does. On completion the software reads the edited screen and generates the communication control language constructs needed to produce the screens.
  - *A communication control language*
    - The screen specifications are written in a programming language (e.g. a 3gl).
- Most generated forms can be *coupled* with a database. This is where the data dictionary comes in.
  - How?
- Some generators allow forms to be database neutral (e.g. for simple input and output screens).



Forms Generator (ii)

- **The more sophisticated generators use the same form specification as a structural template for a *query-by-example* invocation.**

- **Most form generators can automatically create a screen specification for a single table.**
  - It is a good starting point for a more customised design!
  - How form generators tackle multi table forms is crucial for productivity!!
    - **Why?**

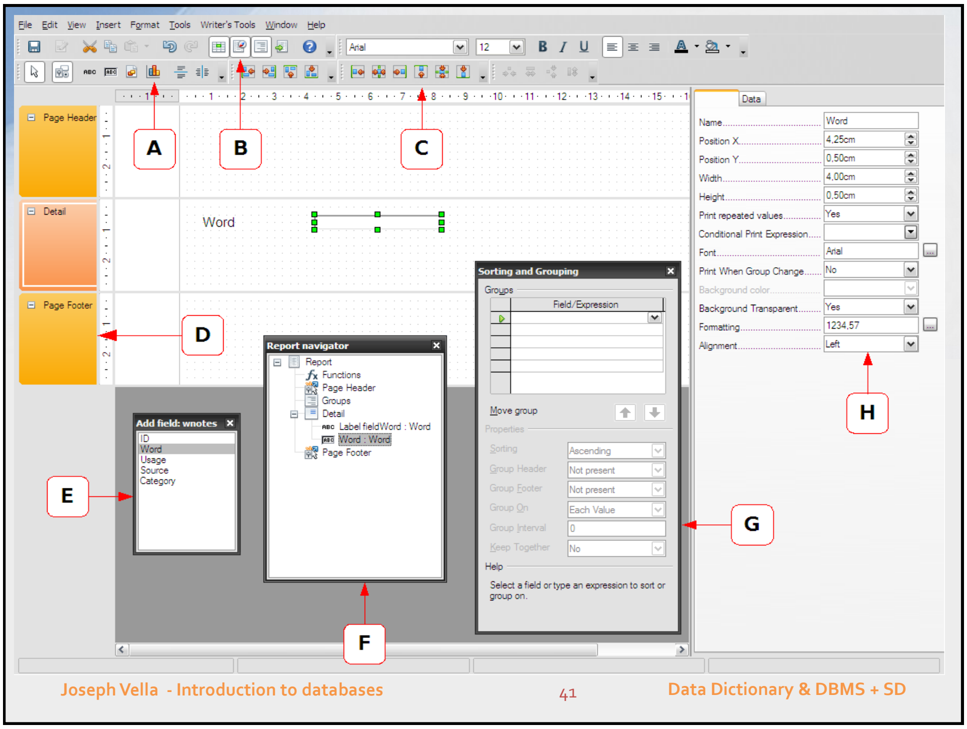
- **Most activities are done by:**
  - declarative constructs; or
  - procedural constructs (e.g. PL/SQL code attached to a form's part); or
  - menu selections & pasting.

- **Other nice features of form generators include:**
  - placing titles and GUI thingies;
  - messages
    - **content and errors;**
  - pre and post data entry actions
    - **declarative or procedural;**
  - pre and post form entry actions;
  - form overlays.

39

Reports

40


File Edit View Insert Format Tools Writer's Tools Window Help
Arial
12
A
B
C
D
E
F
G
H
Page Header
Detail
Page Footer
Word

Add field: wnotes
- ID
- Word
- Usage
- Source
- Category

Report navigator
- Report
  - Functions
  - Page Header
  - Groups
  - Detail
    - Label fieldWord : Word
    - Word : Word
  - Page Footer

Sorting and Grouping

Groups

Field/Expression

Move group

Properties

Sorting	Ascending
Group Header	Not present
Group Footer	Not present
Group On	Each Value
Group Interval	0
Keep Together	No

Help

Select a field or type an expression to sort or group on.

Data

Name	Word
Position X	4,25cm
Position Y	0,50cm
Width	4,00cm
Height	0,50cm
Print repeated values	Yes
Conditional Print Expression	
Font	Arial
Print When Group Change	No
Background color	
Background Transparent	Yes
Formatting	1234,57
Alignment	Left

41

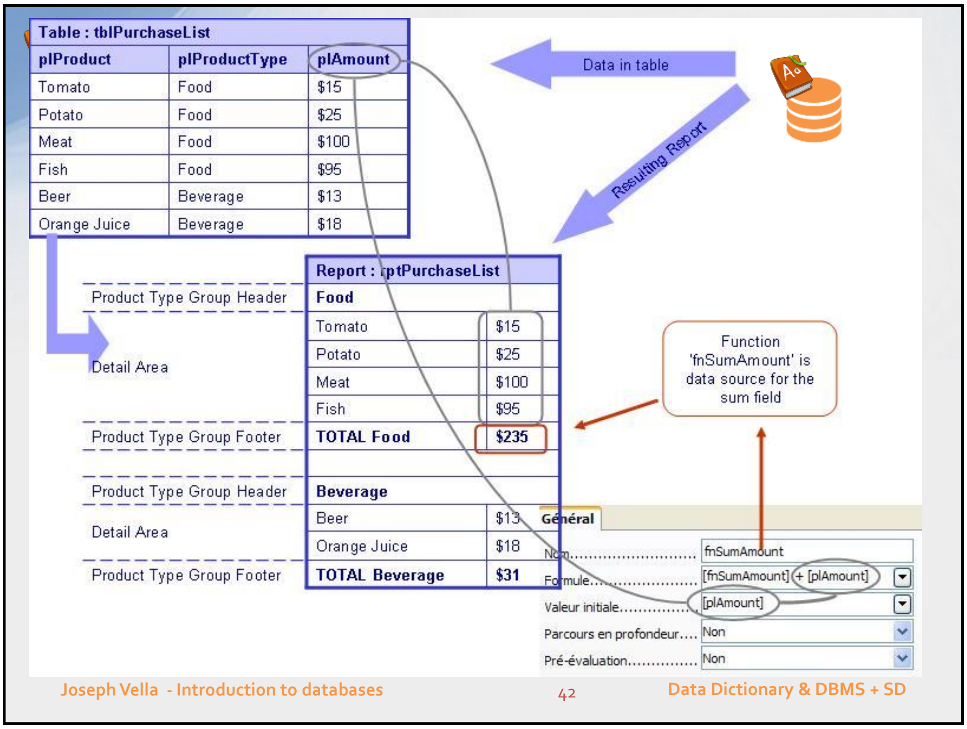


Table : tblPurchaseList

plProduct	plProductType	plAmount
Tomato	Food	$15
Potato	Food	$25
Meat	Food	$100
Fish	Food	$95
Beer	Beverage	$13
Orange Juice	Beverage	$18

Data in table

Resulting Report

Report : rptPurchaseList

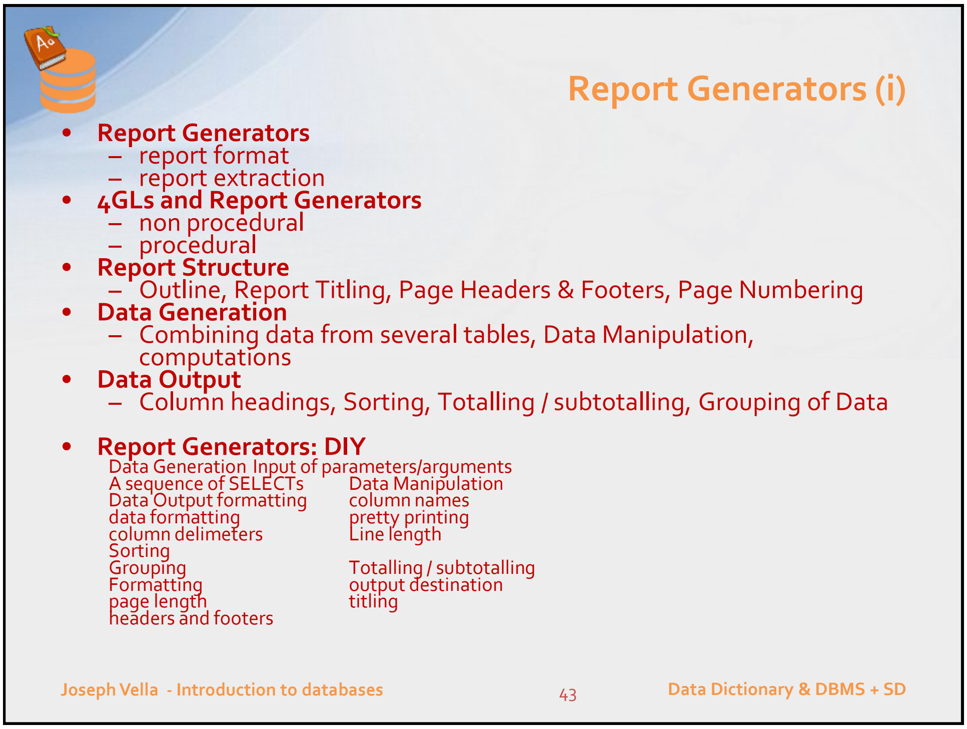
Product Type Group Header	Food	
Detail Area	Tomato	$15
	Potato	$25
	Meat	$100
	Fish	$95
Product Type Group Footer	TOTAL Food	$235
Product Type Group Header	Beverage	
Detail Area	Beer	$13
	Orange Juice	$18
Product Type Group Footer	TOTAL Beverage	$31

Function 'fnSumAmount' is data source for the sum field

Général

Nom	fnSumAmount
Formule	[fnSumAmount] + [plAmount]
Valeur initiale	[plAmount]
Parcours en profondeur	Non
Pré-évaluation	Non

42



Report Generators (i)

- **Report Generators**
  - report format
  - report extraction
- **4GLs and Report Generators**
  - non procedural
  - procedural
- **Report Structure**
  - Outline, Report Titling, Page Headers & Footers, Page Numbering
- **Data Generation**
  - Combining data from several tables, Data Manipulation, computations
- **Data Output**
  - Column headings, Sorting, Totalling / subtotalling, Grouping of Data

- **Report Generators: DIY**

Data Generation Input of parameters/arguments
A sequence of SELECTs
Data Output formatting
data formatting
column delimeters
Sorting
Grouping
Formatting
page length
headers and footers

Data Manipulation
column names
pretty printing
Line length
Totalling / subtotalling
output destination
titling