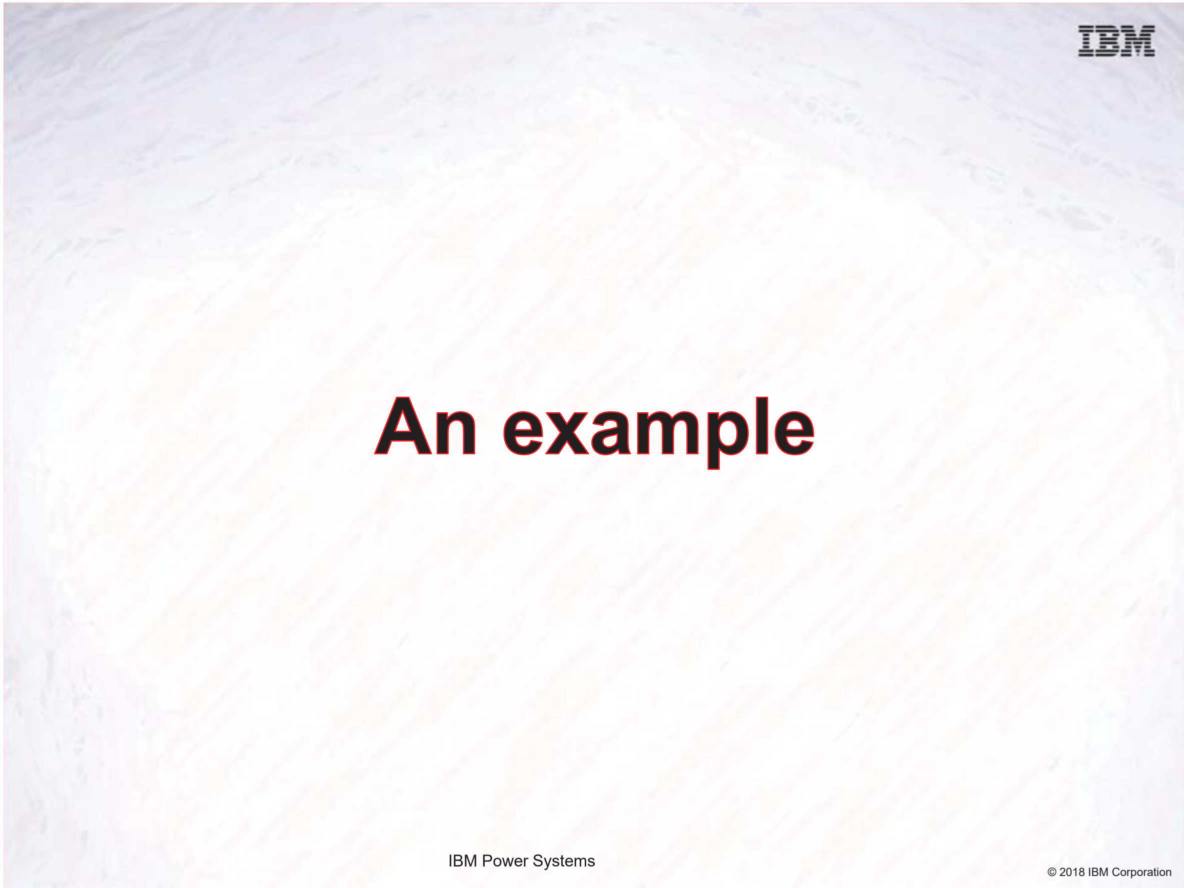
NHMUG
New Hampshire
Midrange User Group

SQL Writing Tips and Techniques

Rob Bestgen
(bestgen@us.ibm.com)
IBM - Db2 for i Consultant

- SQL is a very powerful language for access data
- Utilizing it means leveraging database to do more of the work for you
- Accessing a table is only part of the story
  - It is NOT just a record level access replacement
- Goal: manipulate data to provide **information**



An example

A program with SQL

The day's orders

Get next order

Get customer info

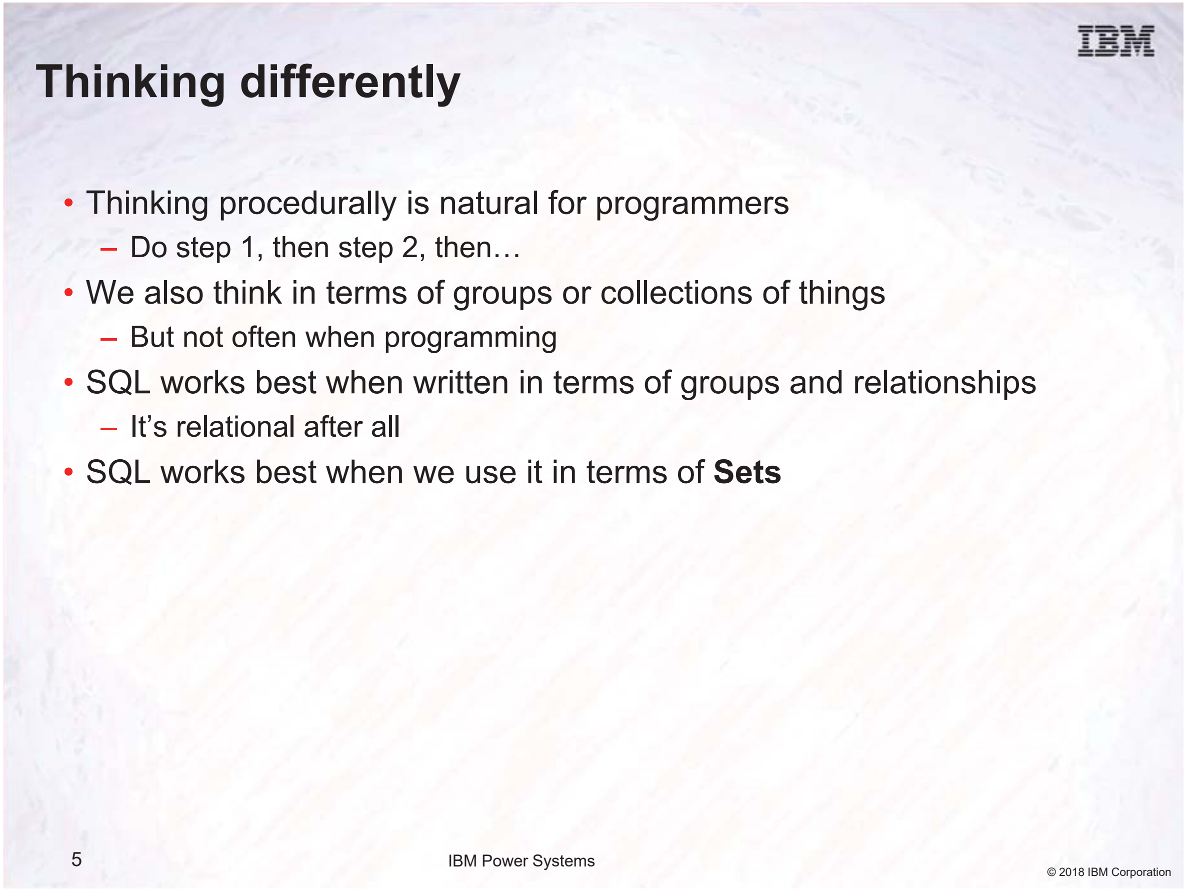
Get product info

Insert into log

Loop to next order

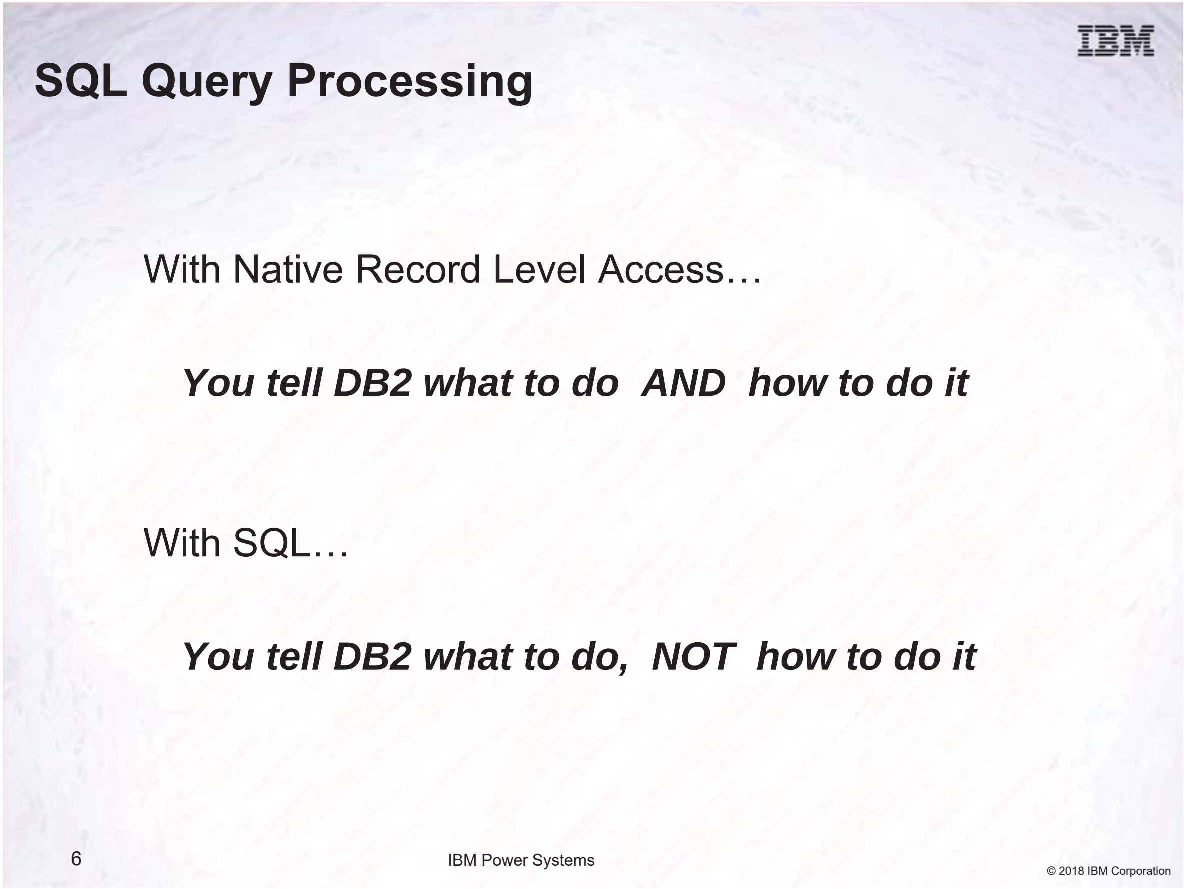
Good or bad?

Why?



Thinking differently

- Thinking procedurally is natural for programmers
  - Do step 1, then step 2, then…
- We also think in terms of groups or collections of things
  - But not often when programming
- SQL works best when written in terms of groups and relationships
  - It’s relational after all
- SQL works best when we use it in terms of **Sets**



SQL Query Processing

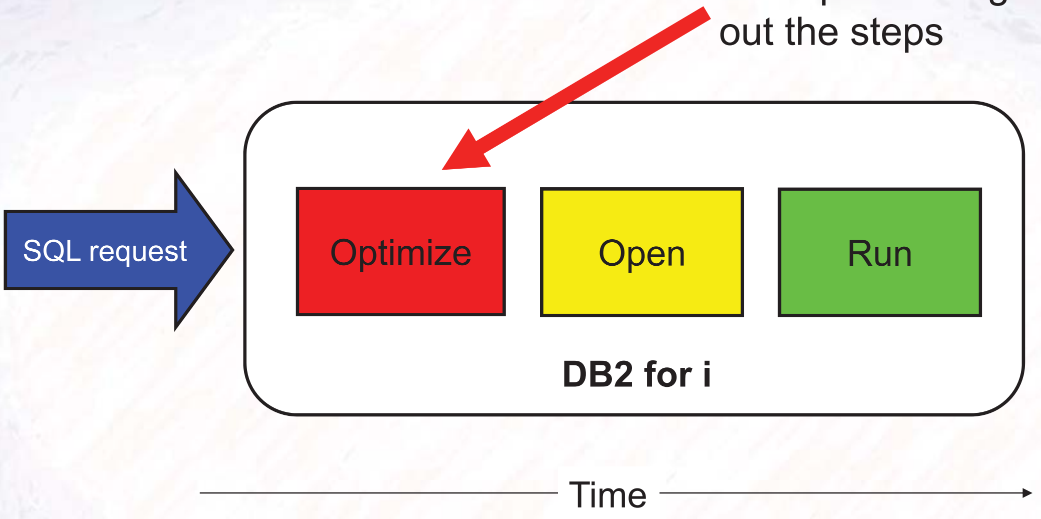
With Native Record Level Access…

You tell DB2 what to do AND how to do it

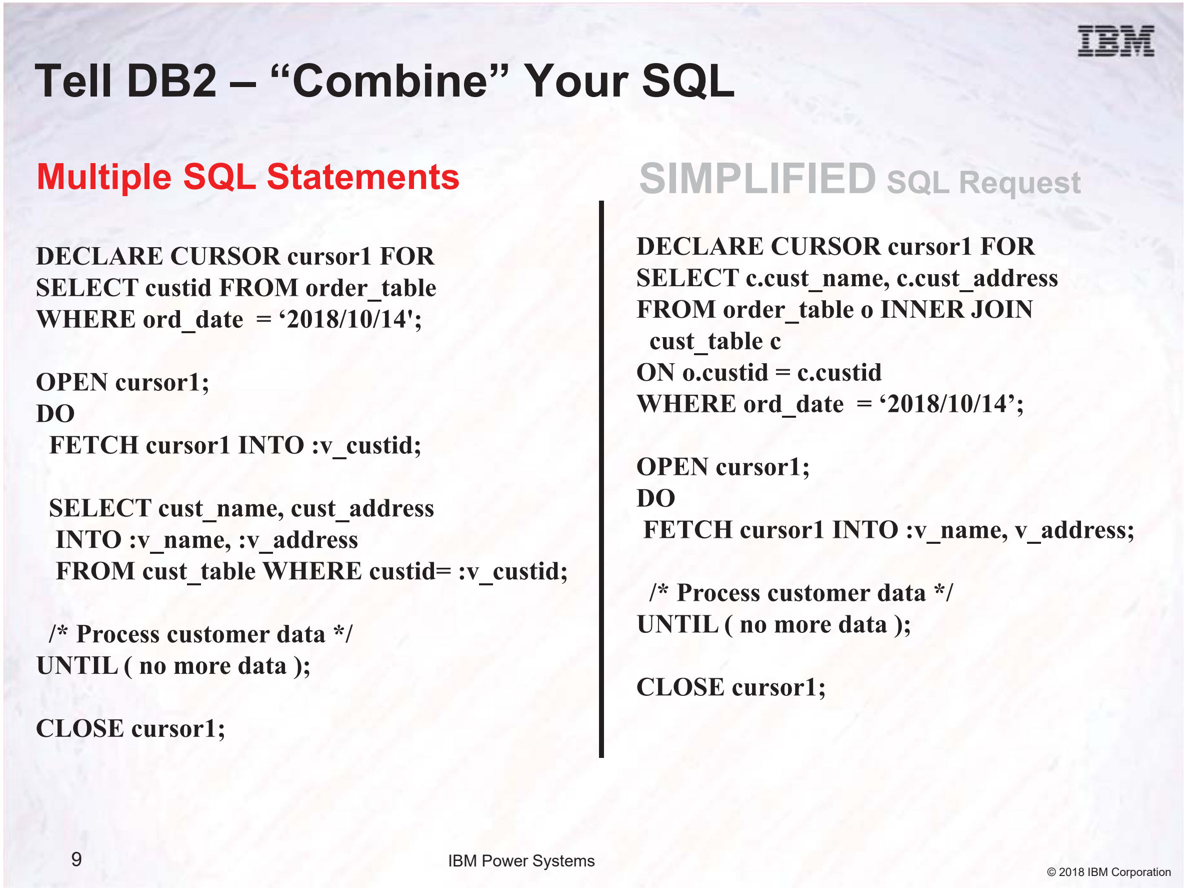
With SQL…

You tell DB2 what to do, NOT how to do it

SQL Query Processing



Combine



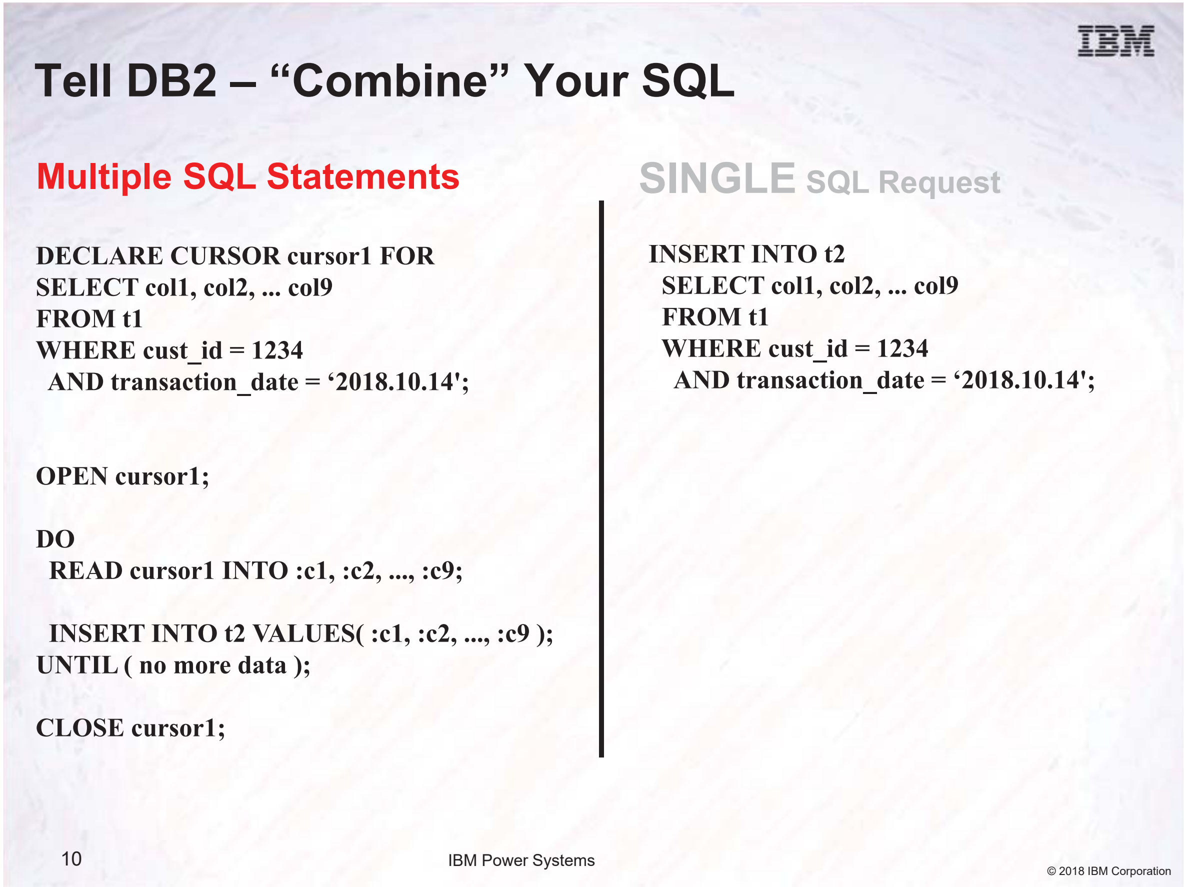
Tell DB2 – “Combine” Your SQL

Multiple SQL Statements

```
DECLARE CURSOR cursor1 FOR
SELECT custid FROM order_table
WHERE ord_date  = ‘2018/10/14';

OPEN cursor1;
DO
  FETCH cursor1 INTO :v_custid;

  SELECT cust_name, cust_address
   INTO :v_name, :v_address
   FROM cust_table WHERE custid= :v_custid;

  /* Process customer data */
UNTIL ( no more data );

CLOSE cursor1;
```

SIMPLIFIED SQL Request

```
DECLARE CURSOR cursor1 FOR
SELECT c.cust_name, c.cust_address
FROM order_table o INNER JOIN
  cust_table c
ON o.custid = c.custid
WHERE ord_date  = ‘2018/10/14’;

OPEN cursor1;
DO
 FETCH cursor1 INTO :v_name, v_address;

  /* Process customer data */
UNTIL ( no more data );

CLOSE cursor1;
```



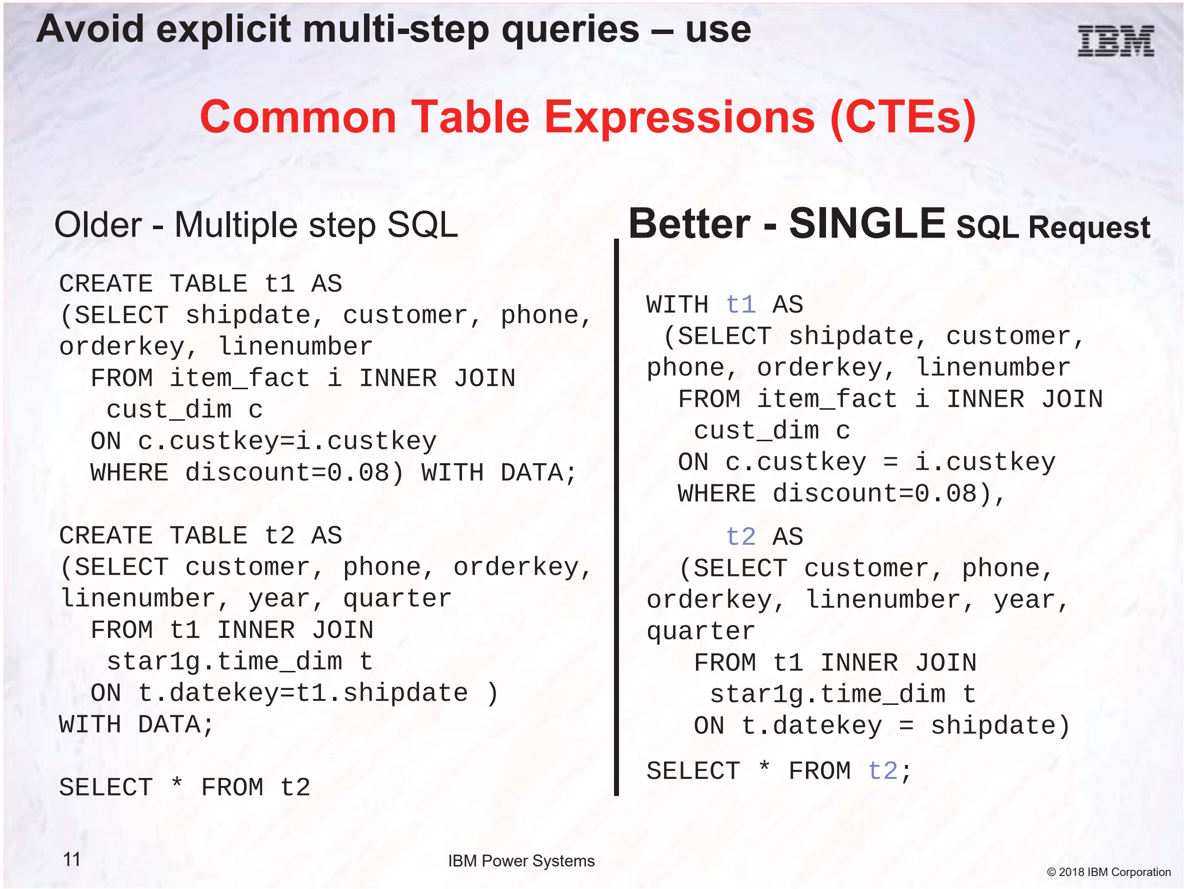
Tell DB2 – “Combine” Your SQL

Multiple SQL Statements

```
DECLARE CURSOR cursor1 FOR
SELECT col1, col2, ... col9
FROM t1
WHERE cust_id = 1234
  AND transaction_date = ‘2018.10.14';


OPEN cursor1;

DO
  READ cursor1 INTO :c1, :c2, ..., :c9;

  INSERT INTO t2 VALUES( :c1, :c2, ..., :c9 );
UNTIL ( no more data );

CLOSE cursor1;
```

SINGLE SQL Request

```
INSERT INTO t2
  SELECT col1, col2, ... col9
  FROM t1
  WHERE cust_id = 1234
    AND transaction_date = ‘2018.10.14';
```



Avoid explicit multi-step queries – use

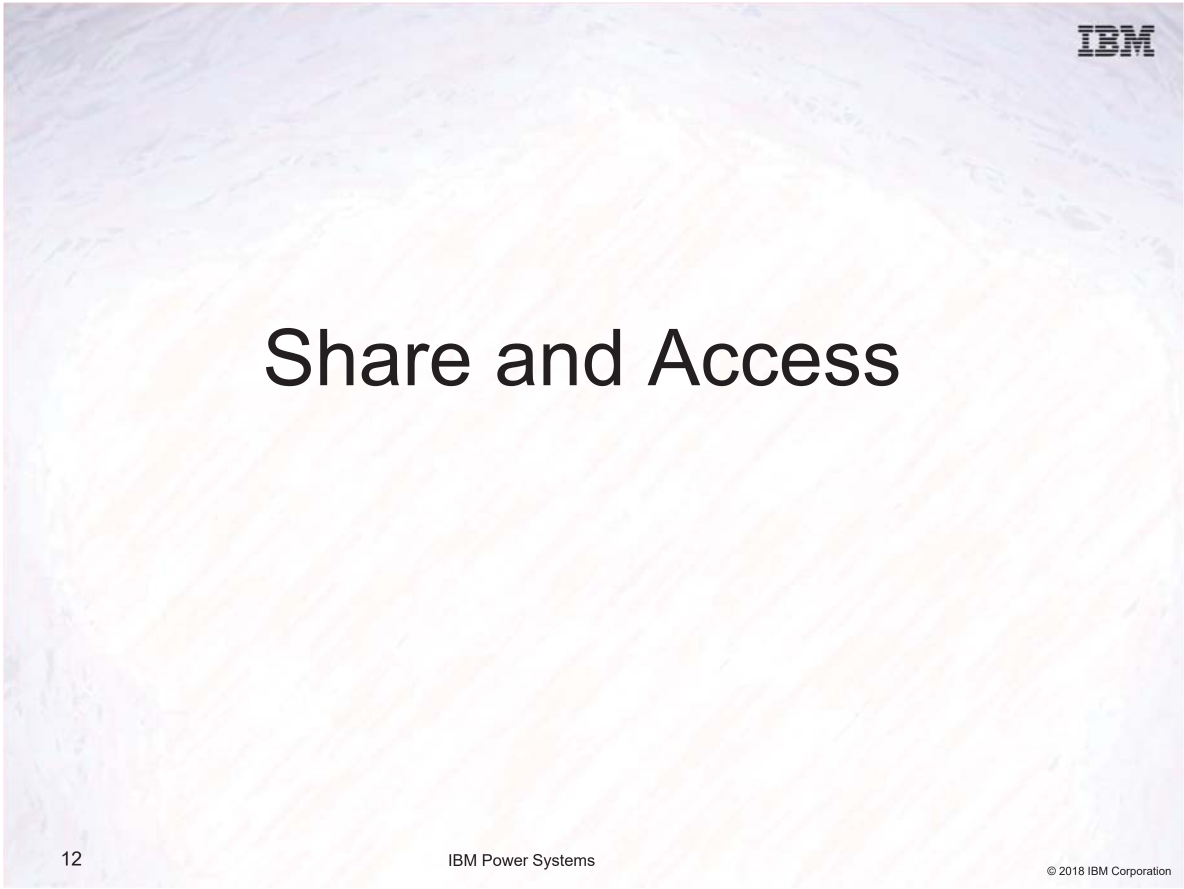
Common Table Expressions (CTEs)

Older - Multiple step SQL

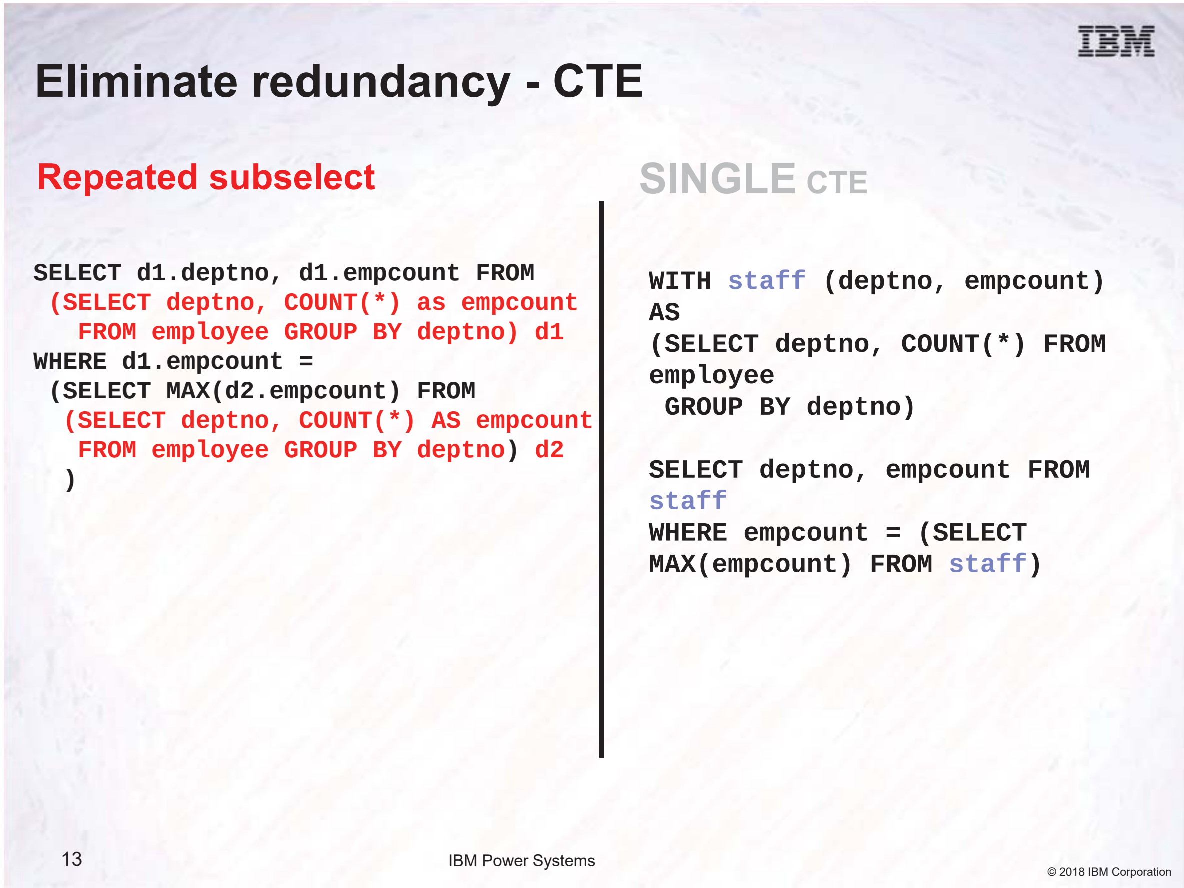
```
CREATE TABLE t1 AS
(SELECT shipdate, customer, phone,
orderkey, linenumber
  FROM item_fact i INNER JOIN
   cust_dim c
  ON c.custkey=i.custkey
  WHERE discount=0.08) WITH DATA;

CREATE TABLE t2 AS
(SELECT customer, phone, orderkey,
linenumber, year, quarter
  FROM t1 INNER JOIN
   star1g.time_dim t
  ON t.datekey=t1.shipdate )
WITH DATA;

SELECT * FROM t2
```

Better - SINGLE SQL Request

```
WITH t1 AS
 (SELECT shipdate, customer,
phone, orderkey, linenumber
  FROM item_fact i INNER JOIN
   cust_dim c
  ON c.custkey = i.custkey
  WHERE discount=0.08),

     t2 AS
  (SELECT customer, phone,
orderkey, linenumber, year,
quarter
   FROM t1 INNER JOIN
    star1g.time_dim t
   ON t.datekey = shipdate)

SELECT * FROM t2;
```



Share and Access



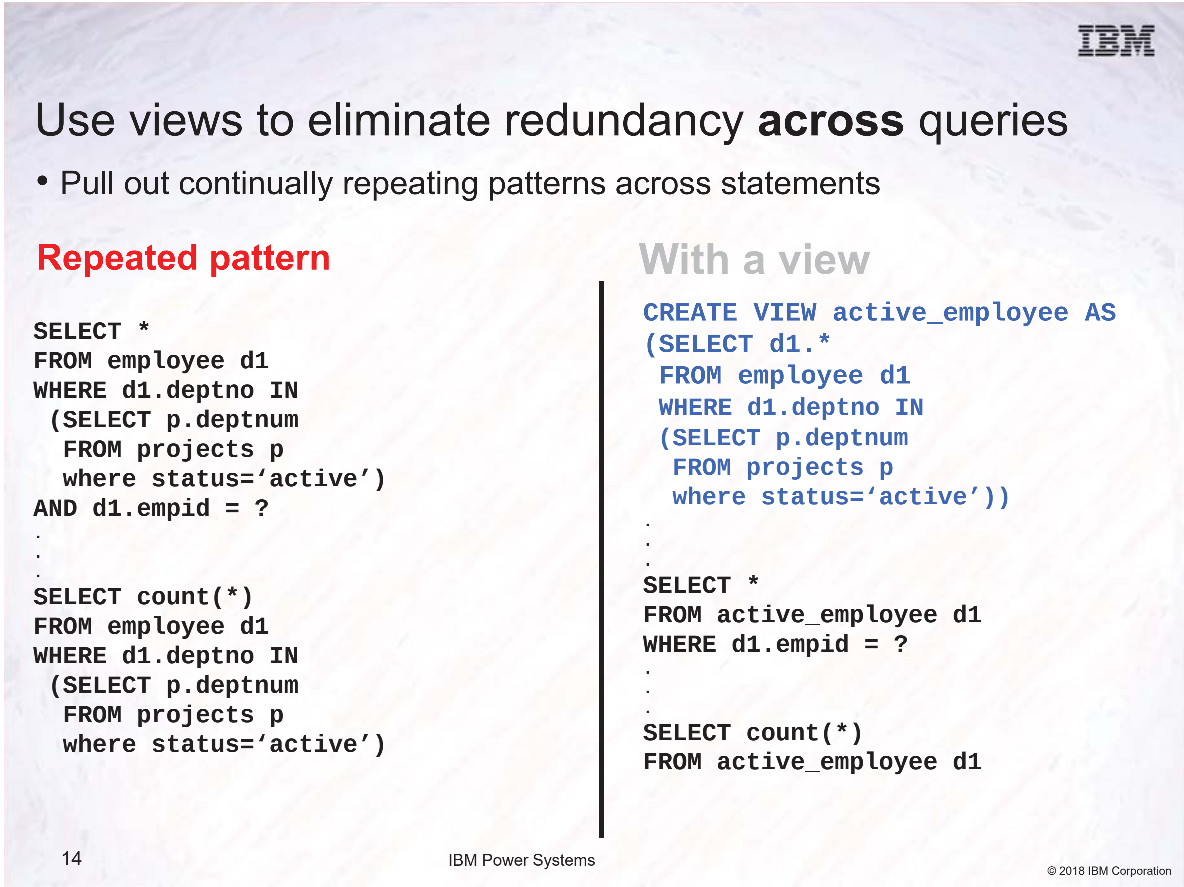
Eliminate redundancy - CTE

Repeated subselect

```
SELECT d1.deptno, d1.empcount FROM
 (SELECT deptno, COUNT(*) as empcount
   FROM employee GROUP BY deptno) d1
WHERE d1.empcount =
 (SELECT MAX(d2.empcount) FROM
  (SELECT deptno, COUNT(*) AS empcount
   FROM employee GROUP BY deptno) d2
  )
```

SINGLE CTE

```
WITH staff (deptno, empcount)
AS
(SELECT deptno, COUNT(*) FROM
employee
 GROUP BY deptno)

SELECT deptno, empcount FROM
staff
WHERE empcount = (SELECT
MAX(empcount) FROM staff)
```



Use views to eliminate redundancy **across** queries

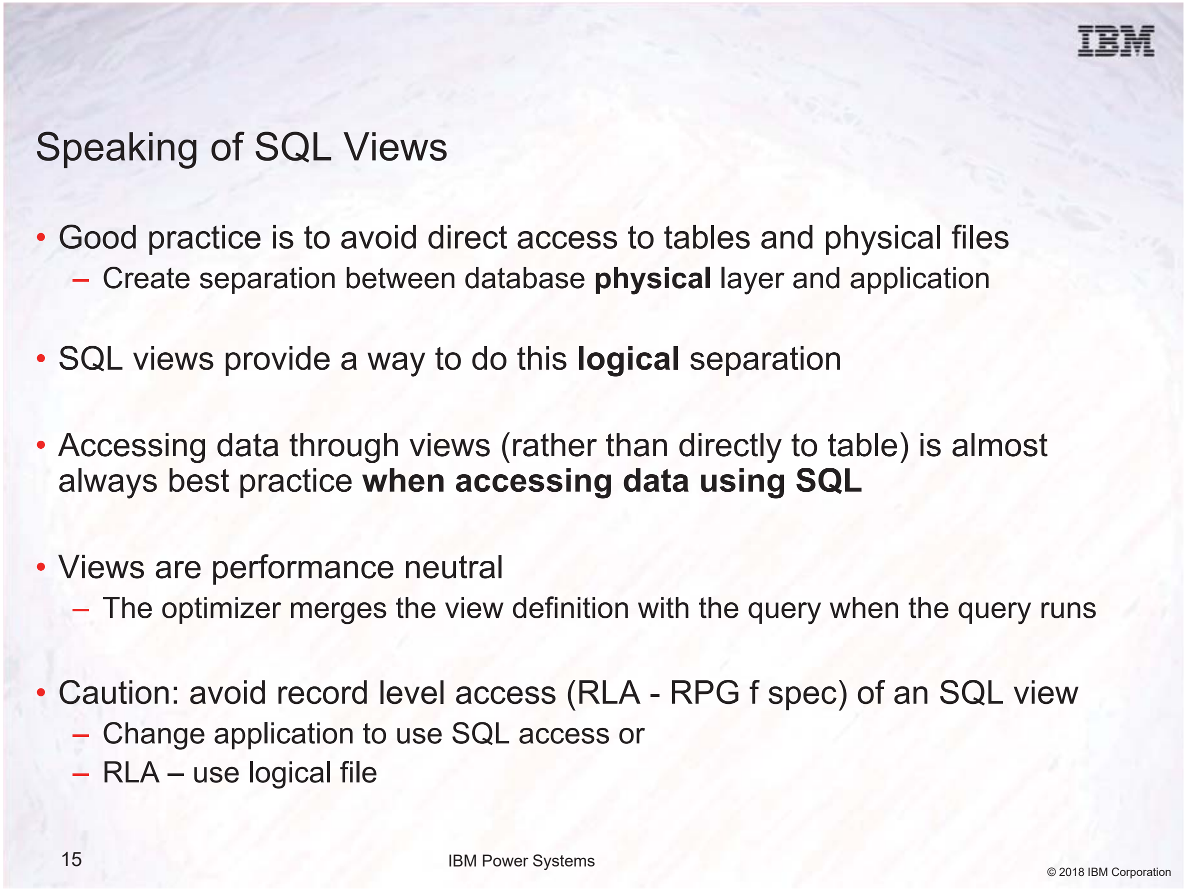
- Pull out continually repeating patterns across statements

Repeated pattern

```
SELECT *
FROM employee d1
WHERE d1.deptno IN
 (SELECT p.deptnum
  FROM projects p
  where status='active')
AND d1.empid = ?
.
.
.
SELECT count(*)
FROM employee d1
WHERE d1.deptno IN
 (SELECT p.deptnum
  FROM projects p
  where status='active')
```

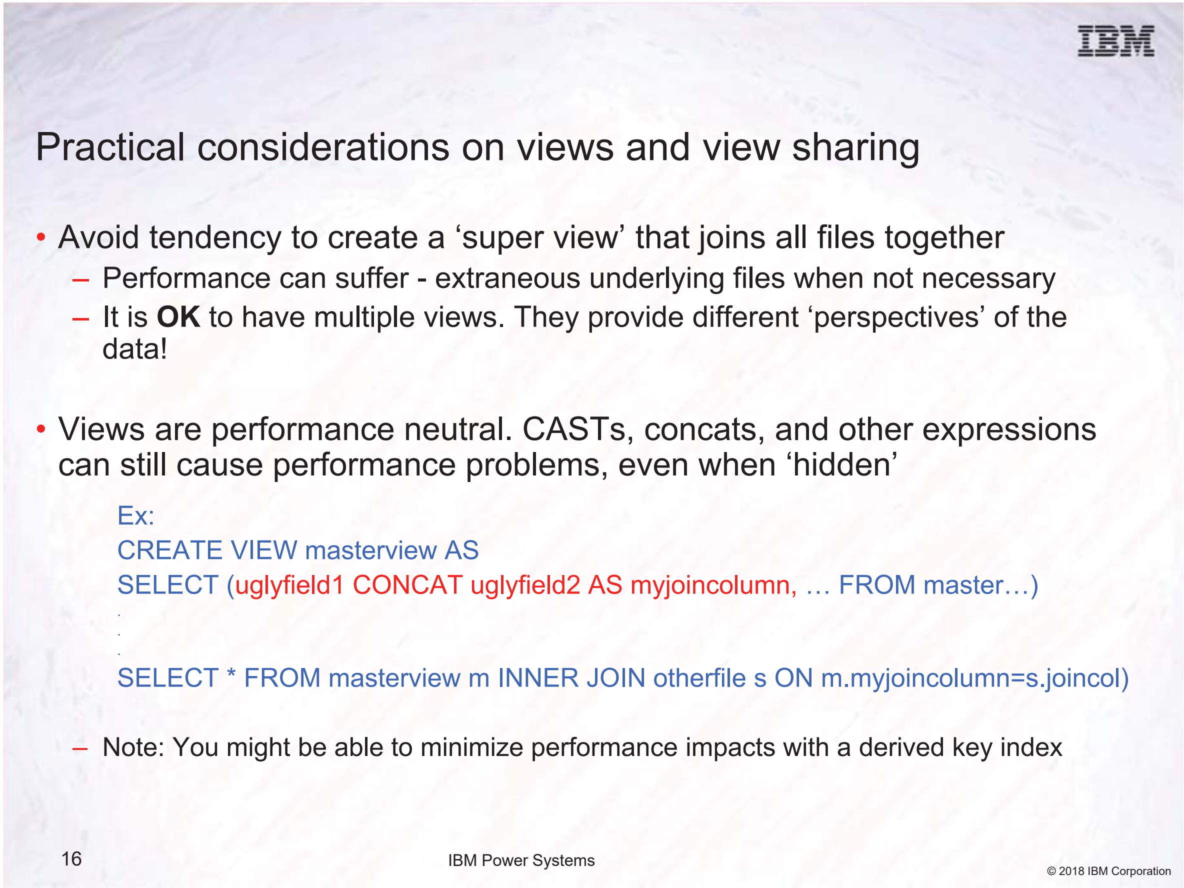
With a view

```
CREATE VIEW active_employee AS
(SELECT d1.*
 FROM employee d1
 WHERE d1.deptno IN
 (SELECT p.deptnum
  FROM projects p
  where status='active'))
.
.
.
SELECT *
FROM active_employee d1
WHERE d1.empid = ?
.
.
.
SELECT count(*)
FROM active_employee d1
```



Speaking of SQL Views

- Good practice is to avoid direct access to tables and physical files
  - Create separation between database **physical** layer and application
- SQL views provide a way to do this **logical** separation
- Accessing data through views (rather than directly to table) is almost always best practice **when accessing data using SQL**
- Views are performance neutral
  - The optimizer merges the view definition with the query when the query runs
- Caution: avoid record level access (RLA - RPG f spec) of an SQL view
  - Change application to use SQL access or
  - RLA – use logical file



Practical considerations on views and view sharing

- Avoid tendency to create a ‘super view’ that joins all files together
  - Performance can suffer - extraneous underlying files when not necessary
  - It is **OK** to have multiple views. They provide different ‘perspectives’ of the data!
- Views are performance neutral. CASTs, concats, and other expressions can still cause performance problems, even when ‘hidden’

Ex:

```
CREATE VIEW masterview AS
SELECT (uglyfield1 CONCAT uglyfield2 AS myjoincolumn, … FROM master…)
.
.
.
SELECT * FROM masterview m INNER JOIN otherfile s ON m.myjoincolumn=s.joincol)
```

  - Note: You might be able to minimize performance impacts with a derived key index

Refactor

- Like all programming languages, it is (too) easy to cut and paste SQL
- Identify when repeats are occurring. Develop a habit of refactoring to use sharing techniques
  - Use CTEs for readability and to eliminate duplicated embedded SELECTs
  - Use (inline) UDFs for common, complex, repeated expressions
  - Create views when SELECTs get complicated, especially if they get repeated
  - Separate code into multiple procedures when repeated code pattern is noticed

Coding Styles

- SQL provides numerous ways to do effectively the same thing
  - Coding styles may be different across developers
- Often it can be personal preference

Different ways, same result

```
SELECT last_name
FROM employee
WHERE status='PT'
AND deptnum IN
 ( SELECT deptno FROM
   location WHERE floornum = 2)
```

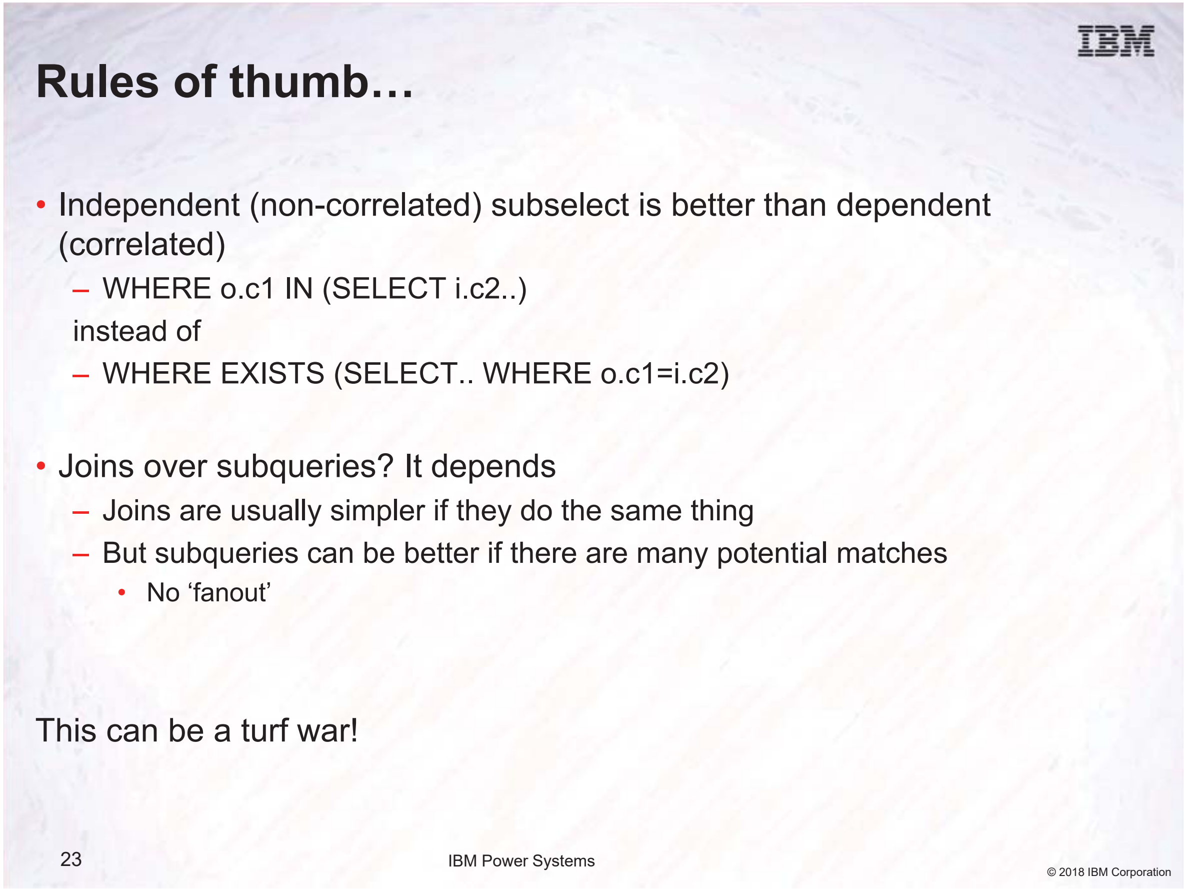
```
SELECT last_name
FROM employee e1 INNER JOIN
 (SELECT DISTINCT deptno FROM
  location WHERE floornum = 2) d1
ON e1.deptnum=d1.deptno
WHERE status='PT'
```

```
SELECT last_name
FROM employee e
WHERE status='PT'
AND EXISTS
  ( SELECT 1 FROM location l
    WHERE floornum = 2
    AND e.deptnum=l.deptno)
```

- So what do you do?
- Many times it is just coding style
- But there are some rules of thumb

Rules of thumb

- Simpler is usually better
  - If you don’t need it, don’t add it
    - Ex: SELECT * FROM t1… but only a few columns are really used
    - Avoid extraneous CASTs
  - Plus, it’s best practice to name the columns, not use SELECT *
- Include just the tables you need
  - Remember the ‘super view’ comment?
  - Having primary/foreign key constraints in place will help the optimizer minimize the negative effects of extraneous tables



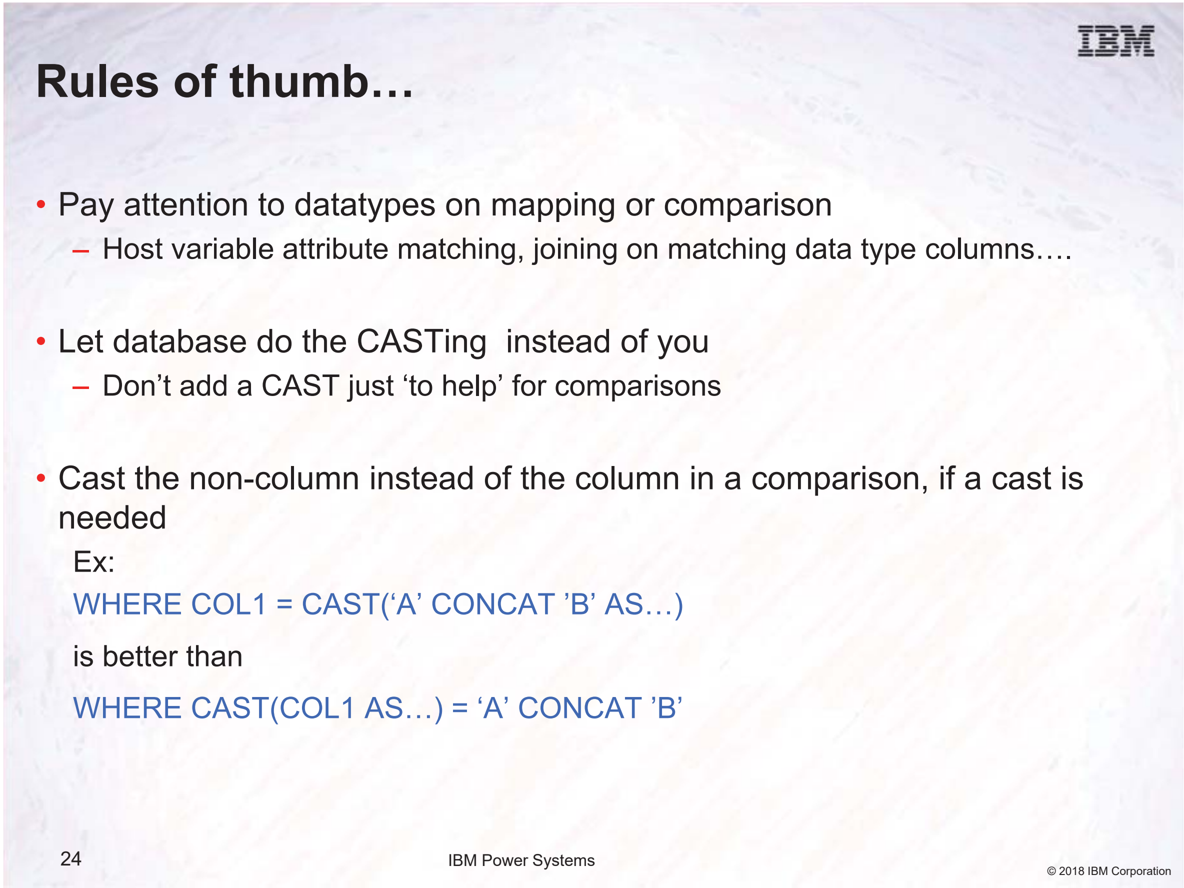
Rules of thumb…

- Independent (non-correlated) subselect is better than dependent (correlated)
  - WHERE o.c1 IN (SELECT i.c2..)

  instead of
  - WHERE EXISTS (SELECT.. WHERE o.c1=i.c2)

- Joins over subqueries? It depends
  - Joins are usually simpler if they do the same thing
  - But subqueries can be better if there are many potential matches
    - No ‘fanout’

This can be a turf war!



Rules of thumb…

- Pay attention to datatypes on mapping or comparison
  - Host variable attribute matching, joining on matching data type columns….

- Let database do the CASTing instead of you
  - Don’t add a CAST just ‘to help’ for comparisons

- Cast the non-column instead of the column in a comparison, if a cast is needed

  Ex:

  WHERE COL1 = CAST(‘A’ CONCAT ’B’ AS…)

  is better than

  WHERE CAST(COL1 AS…) = ‘A’ CONCAT ’B’

Rules of thumb…

- Excessive use of ORs of different columns may indicate an alternate approach is needed
  - there may be a data modeling issue
  - Ex:

```
SELECT * FROM emp INNER JOIN project
ON emp.id = project.eid OR emp.name=project.leadname
```

- Avoid excessive use of NOTs

Helpful Objects

OR REPLACE

OR REPLACE Option for CREATE statements

- Eliminates need for the Drop statement
- Preserves existing object dependencies & privileges!
- Supported objects: Alias, Function, Procedure, Sequence, Trigger, Variable, View, even Table

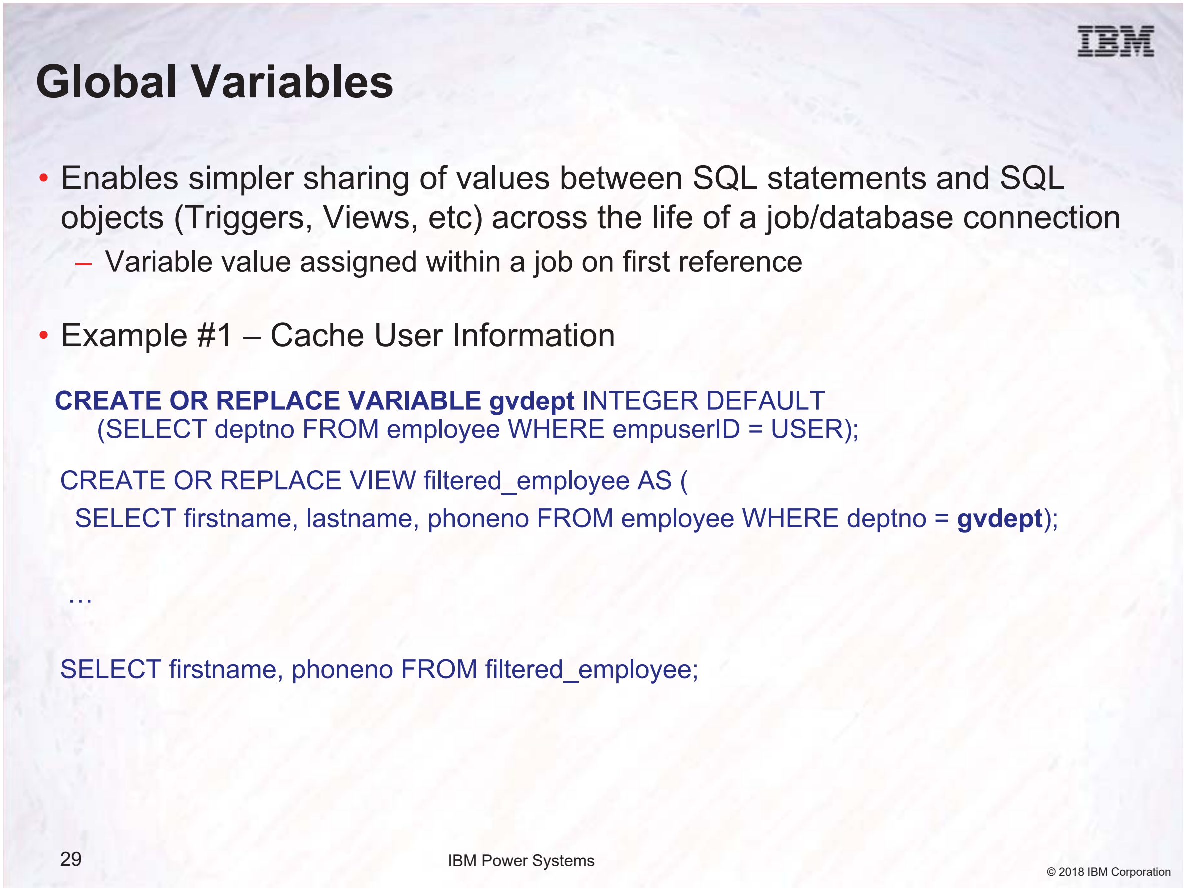
```
FUHDWH#RU#UHSODFH#DOLDV#p|Doldv IRU#vfkhpd1wde4

FUHDWH#RU#UHSODFH#YLHZ#P\bVXUURJDWHý
```

ALIAS

Allows for simpler reference to database files

- Alias is itself a real object on the system
- Great way to reference a particular file member from SQL
- Hides other complexity like three part naming

```
FUHDWH#RU#UHSODFH#DOLDV#FXUPRQWK#IRU#PDLQOLE1VDOHV+PDUFK,
```



Global Variables

- Enables simpler sharing of values between SQL statements and SQL objects (Triggers, Views, etc) across the life of a job/database connection
  - Variable value assigned within a job on first reference
- Example #1 – Cache User Information

```
CREATE OR REPLACE VARIABLE gvdept INTEGER DEFAULT
    (SELECT deptno FROM employee WHERE empuserID = USER);

 CREATE OR REPLACE VIEW filtered_employee AS (
   SELECT firstname, lastname, phoneno FROM employee WHERE deptno = gvdept);

 …

 SELECT firstname, phoneno FROM filtered_employee;
```

Useful on IBM i

RUNSQL CL Command

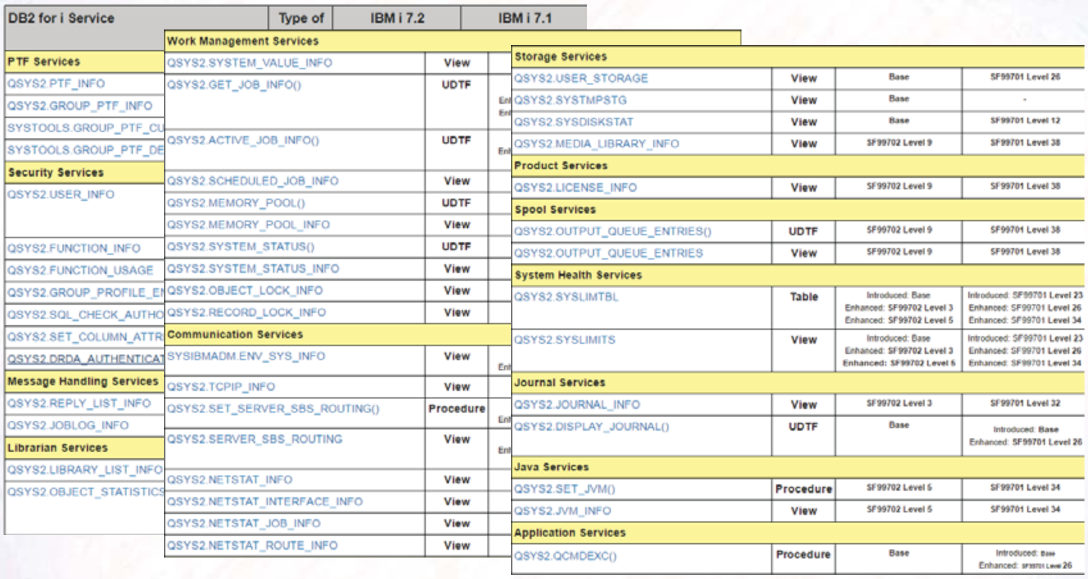
RUNSQL CL command

- Increase adoption of SQL across all interfaces
- Tighter CL program integration than RUNSQLSTM provides
  - SQL can be executed without a source file
  - Limitations:
    - No output support for SELECT statements – temporary tables can be used
    - Error handling limited

```
RUNSQL1: PGM PARM(&LIB)
   DCL &LIB TYPE(*CHAR) LEN(10)
   DCL &SQLSTMT TYPE(*CHAR) LEN(1000)

   CHGVAR VAR(&SQLSTMT) +
     VALUE('INSERT INTO '|| &LIB ||'.TESTABLE VALUES(100,200)')
   RUNSQL SQL(&SQLSTMT) NAMING(*SQL)
ENDSQL1: ENDPGM
```

DB2 for i Services

- Complete listing found on IBM i developerWorks: https://ibm.biz/DB2Services
- Service objects found in QSYS2, unless otherwise noted



DB2 for i Service	Type of	IBM i 7.2	IBM i 7.1
PTF Services			
QSYS2.PTF_INFO			
QSYS2.GROUP_PTF_INFO			
SYSTOOLS.GROUP_PTF_CU			
SYSTOOLS.GROUP_PTF_DE			
Security Services			
QSYS2.USER_INFO			
QSYS2.FUNCTION_INFO			
QSYS2.FUNCTION_USAGE			
QSYS2.GROUP_PROFILE_EI			
QSYS2.SQL_CHECK_AUTHO			
QSYS2.SET_COLUMN_ATTR			
QSYS2.DRDA_AUTHENTICAT			
Message Handling Services			
QSYS2.REPLY_LIST_INFO			
QSYS2.JOBLOG_INFO			
Librarian Services			
QSYS2.LIBRARY_LIST_INFO			
QSYS2.OBJECT_STATISTICS			

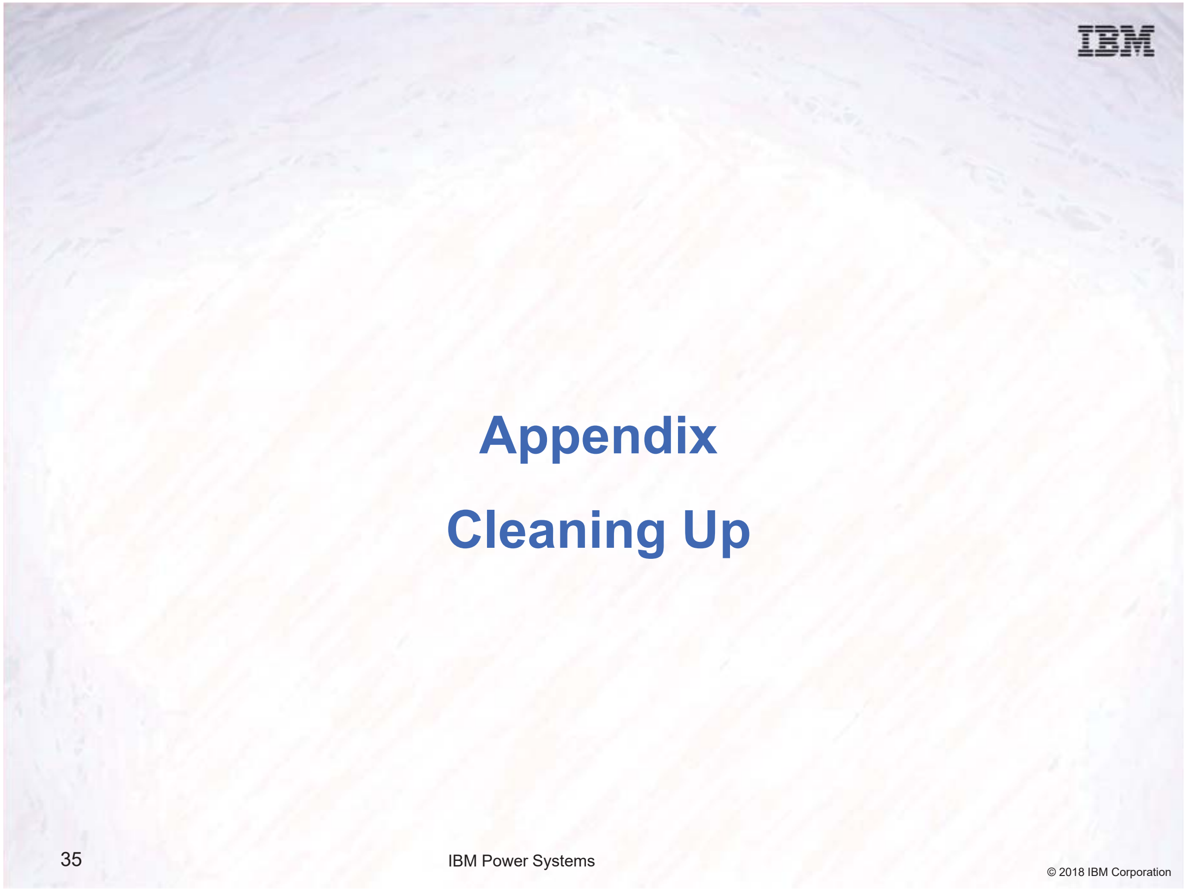
Work Management Services		
QSYS2.SYSTEM_VALUE_INFO	View	
QSYS2.GET_JOB_INFO()	UDTF	Enl Enl
QSYS2.ACTIVE_JOB_INFO()	UDTF	Enl
QSYS2.SCHEDULED_JOB_INFO	View	
QSYS2.MEMORY_POOL()	UDTF	
QSYS2.MEMORY_POOL_INFO	View	
QSYS2.SYSTEM_STATUS()	UDTF	
QSYS2.SYSTEM_STATUS_INFO	View	
QSYS2.OBJECT_LOCK_INFO	View	
QSYS2.RECORD_LOCK_INFO	View	
Communication Services		
SYSIBMADM.ENV_SYS_INFO	View	Enl
QSYS2.TCPIP_INFO	View	
QSYS2.SET_SERVER_SBS_ROUTING()	Procedure	Enl
QSYS2.SERVER_SBS_ROUTING	View	Enl
QSYS2.NETSTAT_INFO	View	
QSYS2.NETSTAT_INTERFACE_INFO	View	
QSYS2.NETSTAT_JOB_INFO	View	
QSYS2.NETSTAT_ROUTE_INFO	View	

Storage Services			
QSYS2.USER_STORAGE	View	Base	SF99701 Level 26
QSYS2.SYSTMPSTG	View	Base	-
QSYS2.SYSDISKSTAT	View	Base	SF99701 Level 12
QSYS2.MEDIA_LIBRARY_INFO	View	SF99702 Level 9	SF99701 Level 38
Product Services			
QSYS2.LICENSE_INFO	View	SF99702 Level 9	SF99701 Level 38
Spool Services			
QSYS2.OUTPUT_QUEUE_ENTRIES()	UDTF	SF99702 Level 9	SF99701 Level 38
QSYS2.OUTPUT_QUEUE_ENTRIES	View	SF99702 Level 9	SF99701 Level 38
System Health Services			
QSYS2.SYSLIMTBL	Table	Introduced: Base Enhanced: SF99702 Level 3 Enhanced: SF99702 Level 5	Introduced: SF99701 Level 23 Enhanced: SF99701 Level 26 Enhanced: SF99701 Level 34
QSYS2.SYSLIMITS	View	Introduced: Base Enhanced: SF99702 Level 3 Enhanced: SF99702 Level 5	Introduced: SF99701 Level 23 Enhanced: SF99701 Level 26 Enhanced: SF99701 Level 34
Journal Services			
QSYS2.JOURNAL_INFO	View	SF99702 Level 3	SF99701 Level 32
QSYS2.DISPLAY_JOURNAL()	UDTF	Base	Introduced: Base Enhanced: SF99701 Level 26
Java Services			
QSYS2.SET_JVM()	Procedure	SF99702 Level 5	SF99701 Level 34
QSYS2.JVM_INFO	View	SF99702 Level 5	SF99701 Level 34
Application Services			
QSYS2.QCMDEXC()	Procedure	Base	Introduced: Base Enhanced: SF99701 Level 26

Summary

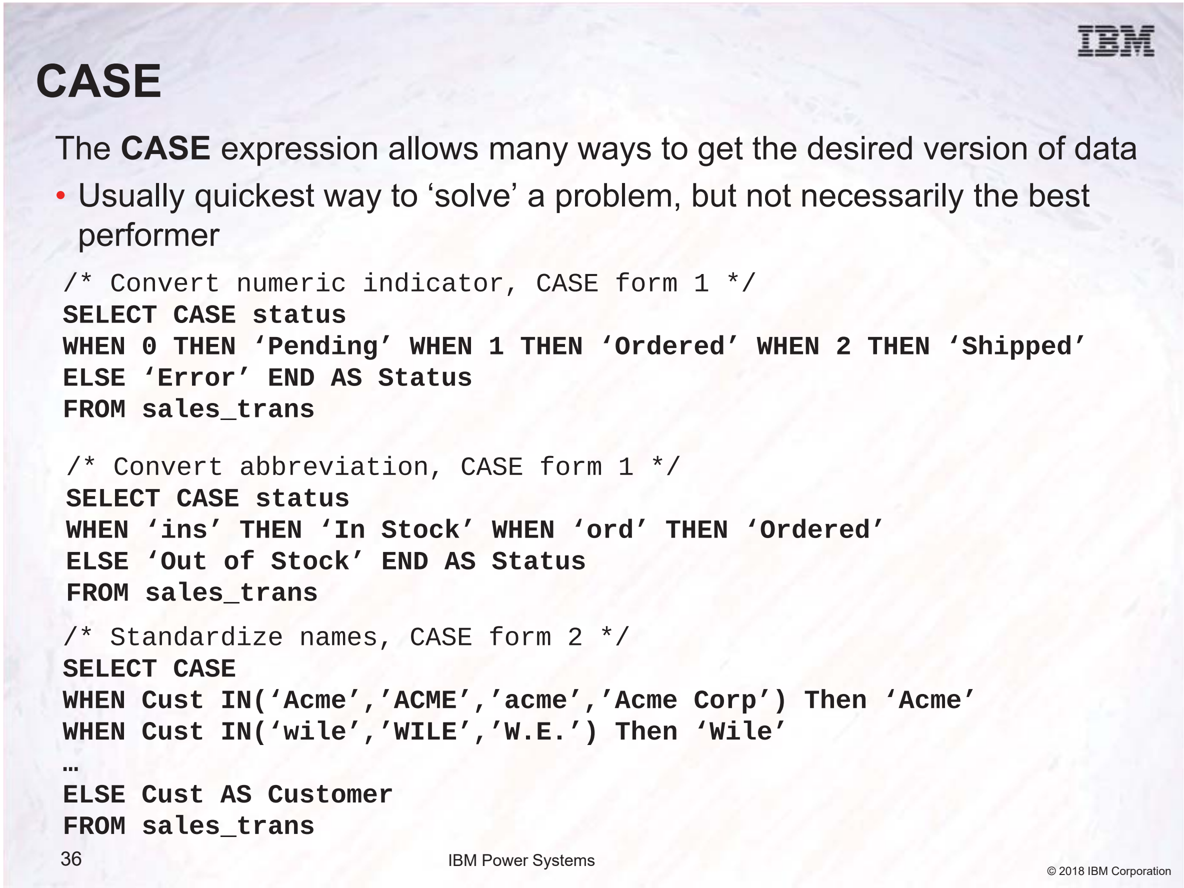
- SQL is a rich language providing many different ways to do ‘the job’
- Understanding and applying the underlying concept behind SQL (set based) helps solve problems in more efficient ways
- Form good SQL habits that work for you while still leveraging the power of SQL

Thank you!



Appendix

Cleaning Up

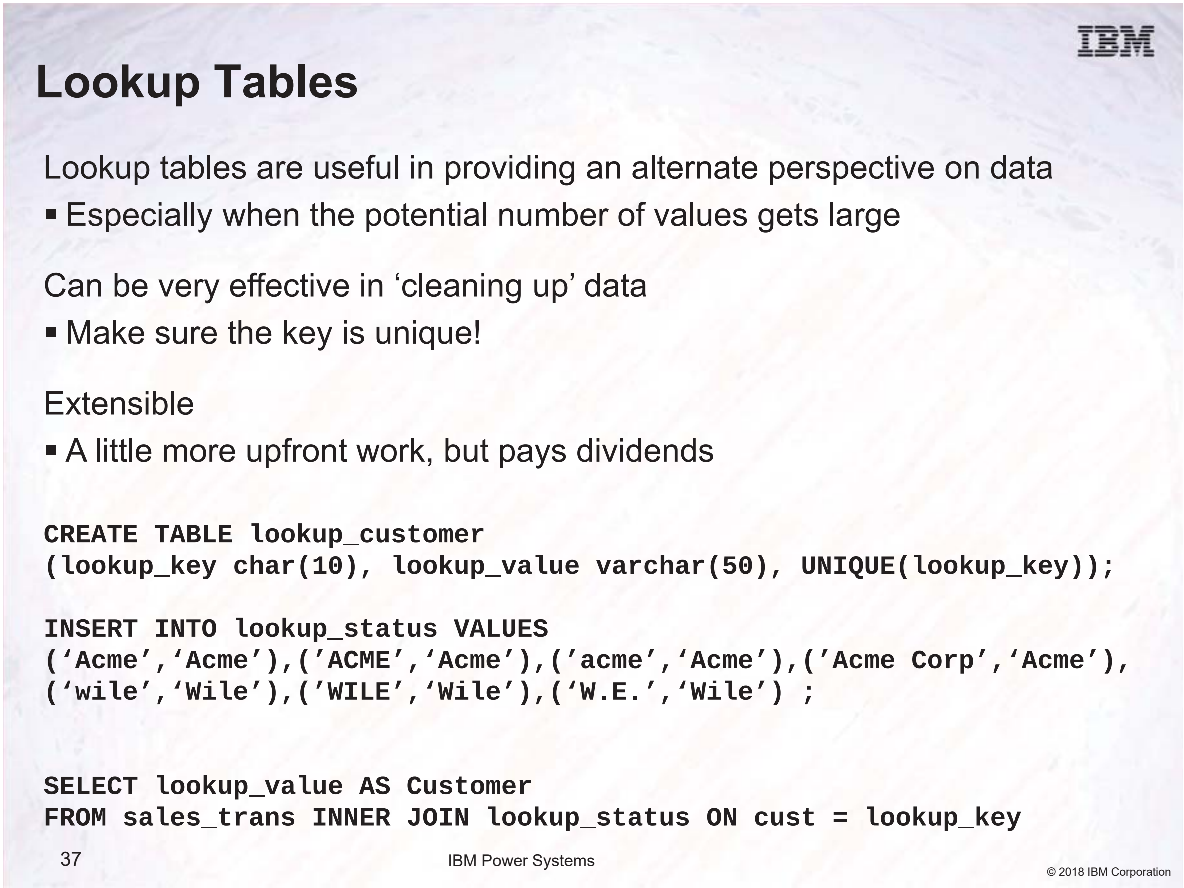


CASE

The **CASE** expression allows many ways to get the desired version of data

- Usually quickest way to ‘solve’ a problem, but not necessarily the best performer

```
/* Convert numeric indicator, CASE form 1 */
SELECT CASE status
WHEN 0 THEN ‘Pending’ WHEN 1 THEN ‘Ordered’ WHEN 2 THEN ‘Shipped’
ELSE ‘Error’ END AS Status
FROM sales_trans

/* Convert abbreviation, CASE form 1 */
SELECT CASE status
WHEN ‘ins’ THEN ‘In Stock’ WHEN ‘ord’ THEN ‘Ordered’
ELSE ‘Out of Stock’ END AS Status
FROM sales_trans

/* Standardize names, CASE form 2 */
SELECT CASE
WHEN Cust IN(‘Acme’,’ACME’,’acme’,’Acme Corp’) Then ‘Acme’
WHEN Cust IN(‘wile’,’WILE’,’W.E.’) Then ‘Wile’
…
ELSE Cust AS Customer
FROM sales_trans
```



Lookup Tables

Lookup tables are useful in providing an alternate perspective on data

- Especially when the potential number of values gets large

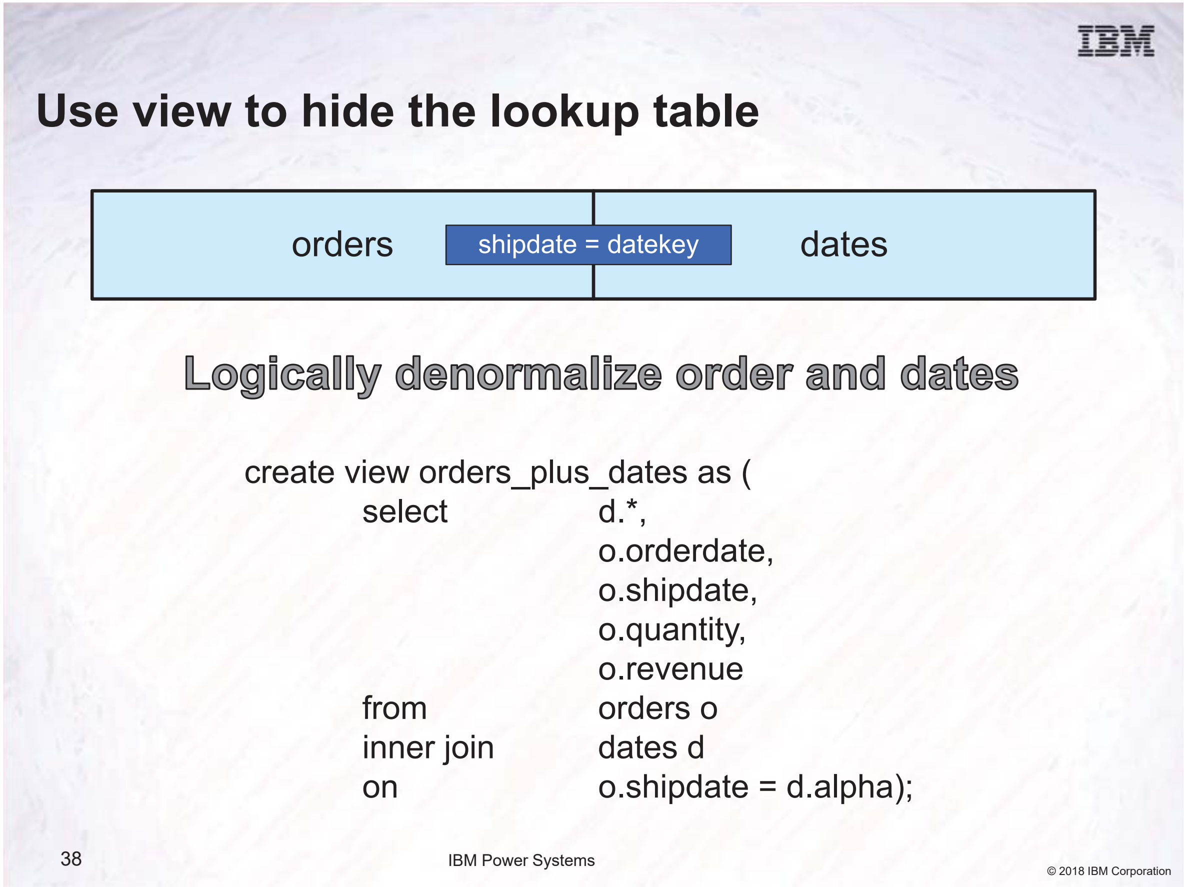
Can be very effective in ‘cleaning up’ data

- Make sure the key is unique!

Extensible

- A little more upfront work, but pays dividends

```
CREATE TABLE lookup_customer
(lookup_key char(10), lookup_value varchar(50), UNIQUE(lookup_key));

INSERT INTO lookup_status VALUES
(‘Acme’,‘Acme’),(’ACME’,‘Acme’),(’acme’,‘Acme’),(’Acme Corp’,‘Acme’),
(‘wile’,‘Wile’),(’WILE’,‘Wile’),(‘W.E.’,‘Wile’) ;


SELECT lookup_value AS Customer
FROM sales_trans INNER JOIN lookup_status ON cust = lookup_key
```

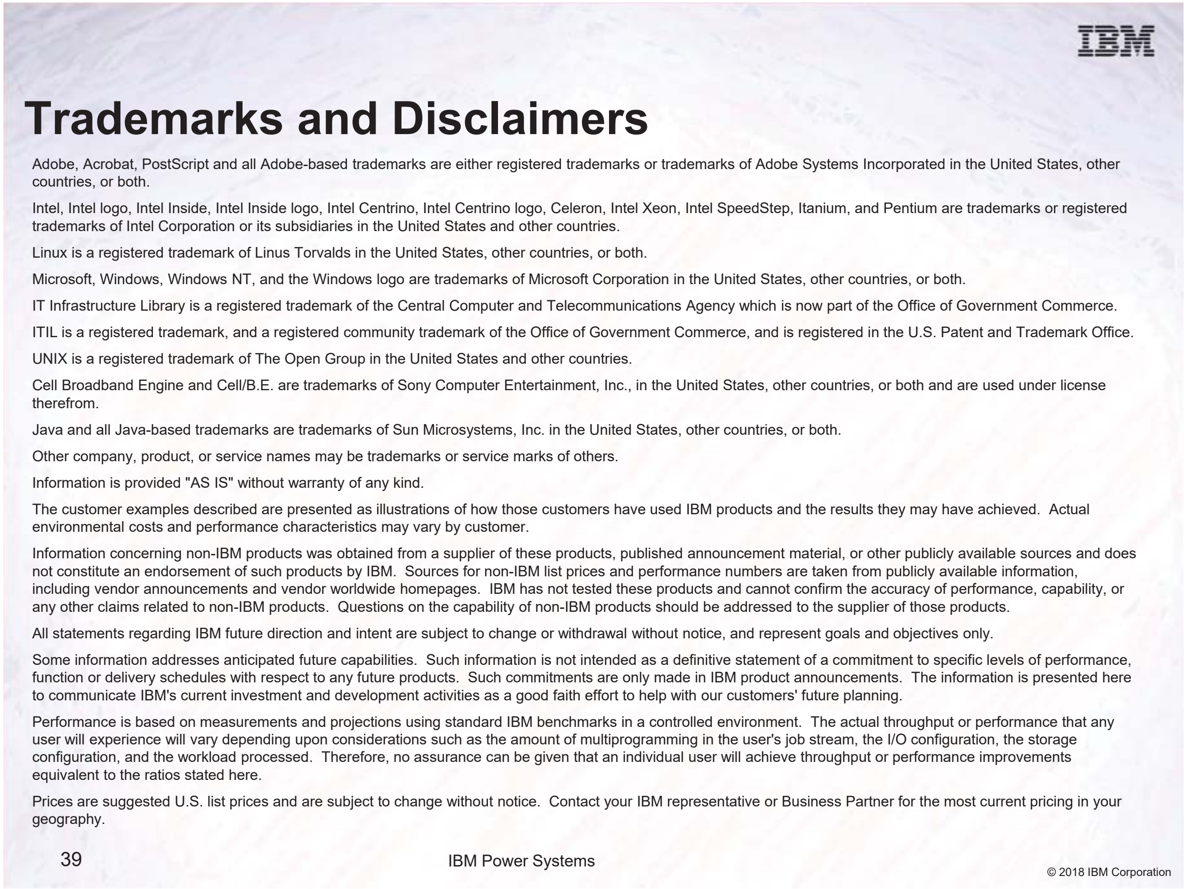


Use view to hide the lookup table

orders
shipdate = datekey
dates

Logically denormalize order and dates

```
create view orders_plus_dates as (
        select          d.*,
                        o.orderdate,
                        o.shipdate,
                        o.quantity,
                        o.revenue
        from            orders o
        inner join      dates d
        on              o.shipdate = d.alpha);
```



Trademarks and Disclaimers

Adobe, Acrobat, PostScript and all Adobe-based trademarks are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Cell Broadband Engine and Cell/B.E. are trademarks of Sony Computer Entertainment, Inc., in the United States, other countries, or both and are used under license therefrom.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Information is provided "AS IS" without warranty of any kind.

The customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer.

Information concerning non-IBM products was obtained from a supplier of these products, published announcement material, or other publicly available sources and does not constitute an endorsement of such products by IBM. Sources for non-IBM list prices and performance numbers are taken from publicly available information, including vendor announcements and vendor worldwide homepages. IBM has not tested these products and cannot confirm the accuracy of performance, capability, or any other claims related to non-IBM products. Questions on the capability of non-IBM products should be addressed to the supplier of those products.

All statements regarding IBM future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Some information addresses anticipated future capabilities. Such information is not intended as a definitive statement of a commitment to specific levels of performance, function or delivery schedules with respect to any future products. Such commitments are only made in IBM product announcements. The information is presented here to communicate IBM's current investment and development activities as a good faith effort to help with our customers' future planning.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Prices are suggested U.S. list prices and are subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.