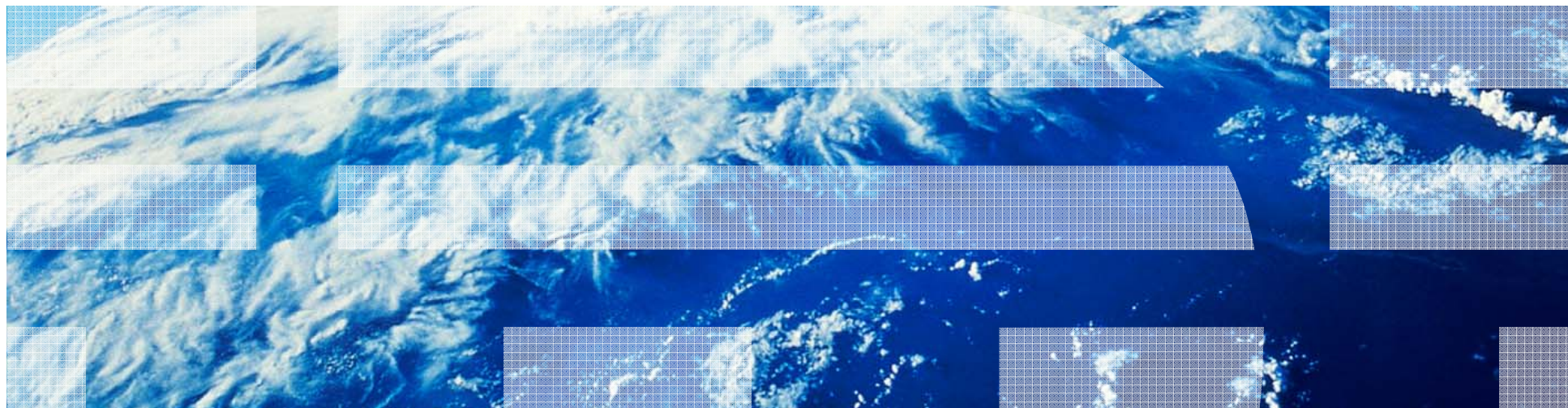


Creating database objects

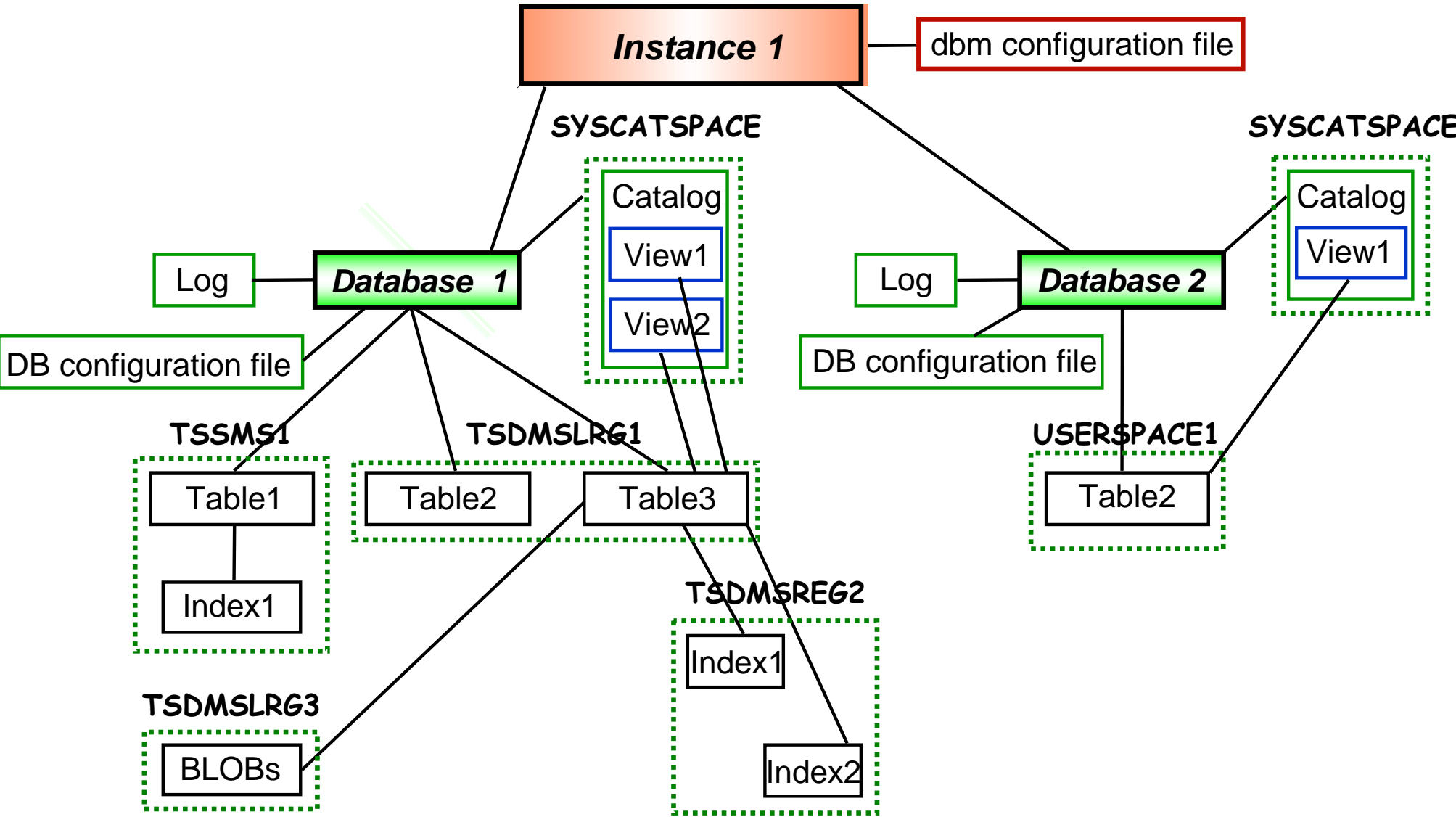


Unit objectives

After completing this unit, you should be able to:

- Describe the DB2 object hierarchy
- Create the following objects:
 - Schema, Table, View, Alias, Index
- Review the use of Temporary Tables
- Explore the use and implementation of Check Constraints, Referential Integrity and Triggers
- Explain the difference between System-period temporal tables and Application-period temporal tables
- List the types of compression available for tables and indexes
- Use the db2look utility to export database structures for future use

DB2 object hierarchy



Create Schema

- A schema is a collection of named objects
- A schema is also a name qualifier

The schema names 'INTERNAL' and 'EXTERNAL' make it easy to distinguish two different SALES tables (INTERNAL.SALES, EXTERNAL.SALES).

- The schema name provides a way to group those objects logically, providing a way to use the same natural name for several objects, and to prevent ambiguous references to those objects.
- Schemas also enable multiple applications to store data in a single database without encountering namespace collisions.
- A schema can contain tables, views, nicknames, triggers, functions, packages, and other objects.
- A schema is itself a database object.
- The schema can be explicitly created using the CREATE SCHEMA statement, with the current user or a specified authorization ID recorded as the schema owner.

```
CREATE SCHEMA PAYROLL  
    AUTHORIZATION DB2ADMIN DATA CAPTURE NONE ;
```

- A schema may also be implicitly created when another object is created, if the user has IMPLICIT_SCHEMA authority
- A schema can be used to set a default DATA CAPTURE option for objects

Set current schema

```
connect to musicdb user Keith;  
select * from employee;
```

- Will select from KEITH.EMPLOYEE

```
set current schema = 'PAYROLL';  
select * from employee;
```

- Will select from PAYROLL.EMPLOYEE

Create Table statement

```
connect to musicdb;
create table artists
(artno          smallint not null,
 name           varchar(50) with default 'abc',
 classification char(1) not null,
 bio            clob(100K) logged,
 picture        blob( 10M) not logged compact)

in dms01
index in dms02
long  in dms03;
```

Application Temporary Tables

- Applications can utilize global temporary tables:
 - Global temporary tables are stored in User temporary table spaces
 - The associated data is deleted when the application connection ends
 - Each connection accesses a private copy of the table, so it only sees data put into the table by that one connection
 - Normal table locking is not needed for global temporary tables since the data is always limited to a single connection
 - Declared Global Temporary table:
 - These are defined during application execution using the DECLARE GLOBAL TEMPORARY TABLE statement
 - No DB2 Catalog information is required
 - The schema for a declared global temporary table is always 'SESSION'
 - Created Global Temporary table:
 - These are defined using a CREATE GLOBAL TEMPORARY TABLE statement with a user selected schema
 - The table and any associated indexes can be created before an application connection starts, but only the catalog definition exists
 - The catalog definition is used when the application references the table
 - Each application connection still works only with the data it stores in the table

Example of a Declared Temporary Table

- A System Administrator creates the user temporary table space

```
CREATE USER TEMPORARY TABLESPACE "USR_TEMP_TS"  
    PAGESIZE 4 K MANAGED BY AUTOMATIC STORAGE  
    BUFFERPOOL IBMDEFAULTBP ;
```

- The application uses SQL statements to declare and access the table

```
DECLARE GLOBAL TEMPORARY TABLE T1  
    LIKE REAL_T1  
    ON COMMIT DELETE ROWS  
    NOT LOGGED  
    IN USR_TEMP_TS;  
INSERT INTO SESSION.T1  
    SELECT * FROM REAL_T1 WHERE DEPTNO=:mydept;
```

- /* do other work on T1 */
- /* when connection ends, table is automatically dropped */

Example of a Created Temporary Table

- A System Administrator creates the user temporary table space and defines a global temporary table and indexes

```
CREATE USER TEMPORARY TABLESPACE "USR_TEMP_TS2"  
    PAGESIZE 4 K MANAGED BY AUTOMATIC STORAGE ;
```

```
CREATE GLOBAL TEMPORARY TABLE APP1.DEPARTMENT  
    LIKE PROD.DEPARTMENT  
    ON COMMIT DELETE ROWS  
    NOT LOGGED IN USR_TEMP_TS2;
```

```
CREATE INDEX APP1.DEPTIX1 ON APP1.DEPARTMENT (DEPTNO);
```

- The application uses SQL statements to reference the temporary table; no DECLARE is needed

```
INSERT INTO APP1.DEPARTMENT  
    SELECT * FROM PROD.DEPARTMENT WHERE DEPTNO=:mydept;
```

```
SELECT * FROM APP1.DEPARTMENT WHERE LASTNAME = 'STOPFER'
```

Table partitioning

- Data organization scheme in which table data is divided across multiple storage objects called data partitions or ranges:
 - Each data partition is stored separately
 - These storage objects can be in different table spaces, in the same table space, or a combination of both
- Benefits:
 - Easier roll-in and roll-out of table data
 - Allows large data roll-in (ATTACH) or roll-out (DETACH) with a minimal impact to table availability for applications
 - Supports very large tables
 - Indexes can be either partitioned (local) or non-partitioned (global)
 - Table and Index scans can use partition elimination when access includes predicates for the defined ranges
 - Different ranges can be assigned to table spaces in different storage groups for current data versus less used historical data

Example of a range partitioned table

- The PARTITION BY RANGE clause defines a set of data ranges

```
CREATE TABLE PARTTAB.HISTORYPART ( ACCT_ID INTEGER NOT NULL ,
    TELLER_ID SMALLINT NOT NULL ,
    BRANCH_ID SMALLINT NOT NULL ,
    BALANCE DECIMAL(15,2) NOT NULL ,
    ..... .
    TEMP CHAR(6) NOT NULL )
PARTITION BY RANGE (BRANCH_ID)
(STARTING FROM (1) ENDING (20) IN TSHISTP1 INDEX IN TSHISTI1 ,
  STARTING FROM (21) ENDING (40) IN TSHISTP2 INDEX IN TSHISTI2 ,
  STARTING FROM (41) ENDING (60) IN TSHISTP3 INDEX IN TSHISTI3 ,
  STARTING FROM (61) ENDING (80) IN TSHISTP4 INDEX IN TSHISTI4 ) ;
```

```
CREATE INDEX PARTTAB.HISTPIX1 ON PARTTAB.HISTORYPART (TELLER_ID)
    PARTITIONED ;
```

```
CREATE INDEX PARTTAB.HISTPIX2 ON PARTTAB.HISTORYPART (BRANCH_ID)
    PARTITIONED ;
```

- In this example, the data objects and index objects for each data range are stored in different table spaces
- The table spaces used must be defined with the same options, such as type of management, extent size and page size

Create View statement

```
CONNECT TO TESTDB;
```

```
CREATE VIEW EMPSALARY
```

```
    AS SELECT EMPNO, EMPNAME, SALARY
```

```
    FROM PAYROLL, PERSONNEL
```

```
    WHERE EMPNO=EMPNUMB AND SALARY > 30000.00;
```

```
SELECT * FROM EMPSALARY
```

EMPNO	EMPNAME	SALARY
-----	-----	-----
10	John Smith	1000000.00
20	Jane Johnson	300000.00
30	Robert Appleton	250000.00
...		

Create Alias statement

- Cannot be the same as an existing table, view, or alias

**To create an alias of ADMIN.MUSICIANS for the table
ADMIN.ARTISTS**

```
CREATE ALIAS ADMIN.MUSICIANS FOR ADMIN.ARTISTS;
```

**To create a public alias called TABS for the catalog view
SYSCAT.TABLES.**

```
CREATE PUBLIC ALIAS TABS FOR SYSCAT.TABLES
```

Create Index statements

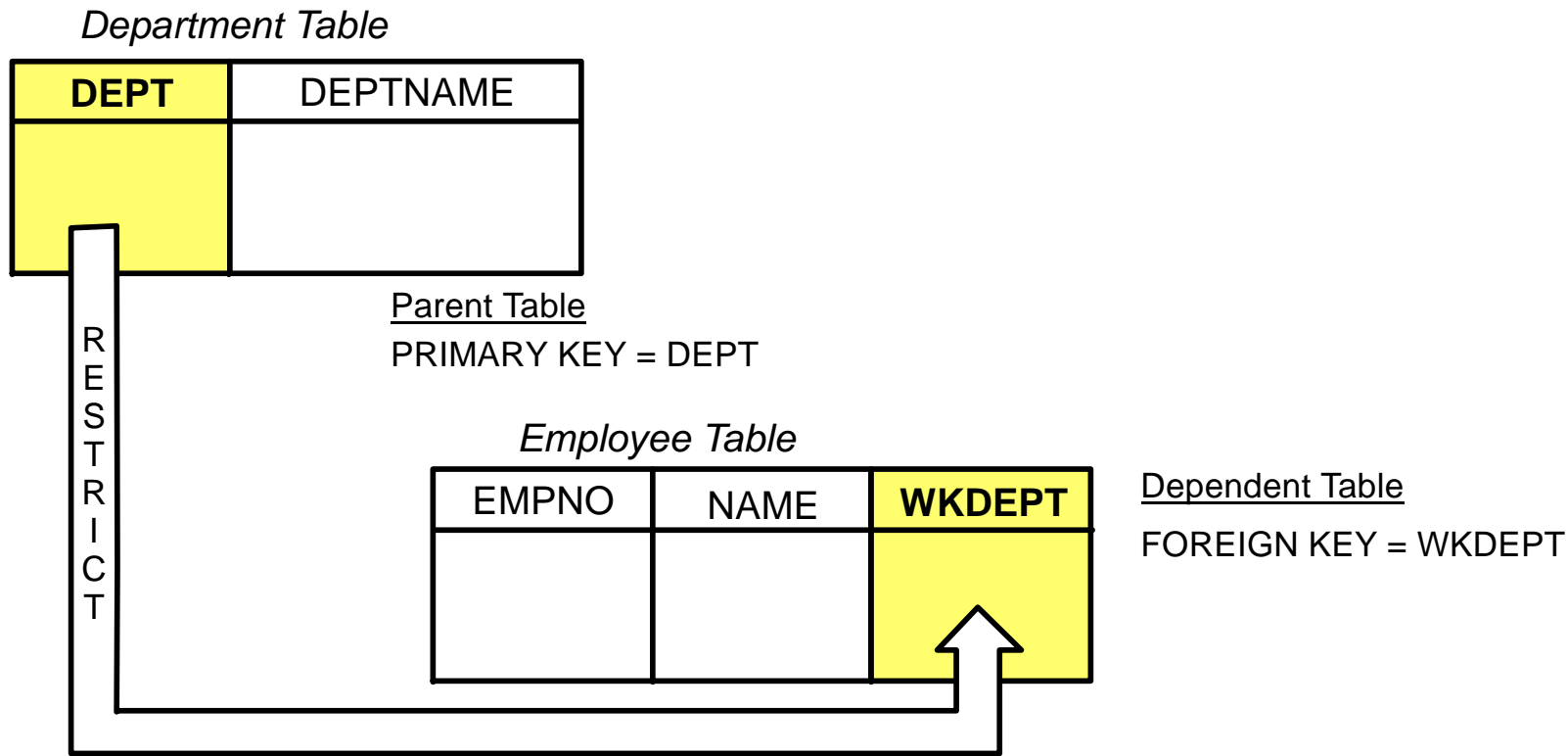
```
create unique index dba1.empno on dba1.employee  
  (empno asc)  
  pctfree 10  
  minpctused 10  
  allow reverse scans  
  page split symmetric  
  collect sampled detailed statistics ;
```

```
create unique index itemno on albums (itemno) ;
```

```
create index item on stock (itemno) cluster ;
```

```
create unique index empidx on employee (empno)  
  include (lastname, firstname) ;
```

Overview of Referential Integrity



- Place constraints between tables
- Constraints specified with Create and Alter table statements
- Database services enforce constraints: inserts, updates, deletes
- Removes burden of constraint checking from application programs

Referential Integrity: CREATE TABLE statement

```
CREATE TABLE DEPARTMENT
```

```
    (DEPTNO      CHAR(3)      NOT NULL,  
     DEPTNAME    VARCHAR(29)  NOT NULL,  
     MGRNO       CHAR(6),  
     ADMRDEPT    CHAR(3)      NOT NULL,  
     LOCATION    CHAR(16),  
     PRIMARY KEY (DEPTNO))
```

```
IN RESOURCE
```

```
CREATE TABLE EMPLOYEE
```

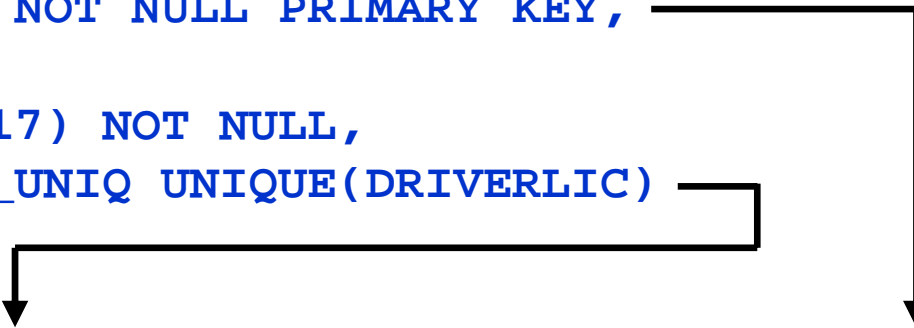
```
    (EMPNO       CHAR(6)      NOT NULL PRIMARY KEY,  
     FIRSTNAME   VARCHAR(12)  NOT NULL,  
     LASTNAME    VARCHAR(15)  NOT NULL,  
     WORKDEPT    CHAR(3),  
     PHONENO     CHAR(4),  
     PHOTO       BLOB(10m)    NOT NULL,  
     FOREIGN KEY DEPT (WORKDEPT)  
     REFERENCES DEPARTMENT ON DELETE NO ACTION)
```

```
IN RESOURCE
```


Unique Key considerations

- Multiple keys in one table can be Foreign Key targets

```
CREATE TABLE PAY.EMPTAB  
  (EMPNO SMALLINT NOT NULL PRIMARY KEY,  
   NAME CHAR(20),  
   DRIVERLIC CHAR(17) NOT NULL,  
   CONSTRAINT DRIV_UNIQ UNIQUE(DRIVERLIC)  
 )IN TBSP1 ;
```

A diagram with two arrows. One arrow starts from the 'PRIMARY KEY' text in the SQL code and points down to the text 'Unique indexes PAY.DRIV_UNIQ and SYSIBM.yymmddhhmmssxxx created Columns must be NOT NULL'. The other arrow starts from the 'UNIQUE(DRIVERLIC)' text in the SQL code and points down to the same text.

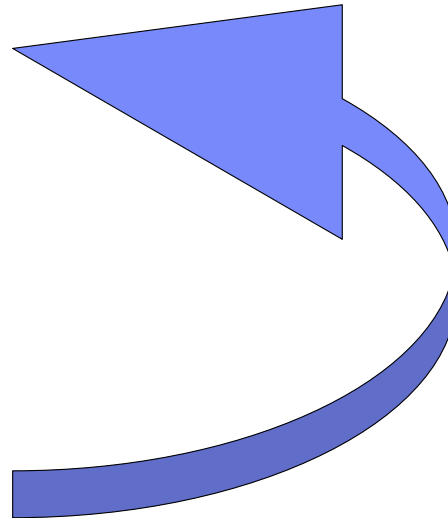
Unique indexes PAY.DRIV_UNIQ and SYSIBM.yymmddhhmmssxxx
created Columns must be NOT NULL

- Deferral of unique checking until end of statement processing
 - UPDATE EMPTAB SET EMPNO=EMPNO + 1

Check Constraints: Definition

```
CREATE TABLE SPEED_LIMITS
(ROUTE_NUM    SMALLINT,
 CANADA_SL    INTEGER NOT NULL,
 US_SL        INTEGER NOT NULL
CHECK (US_SL <=65) ) ;
```

```
ALTER TABLE SPEED_LIMITS
ADD
CONSTRAINT SPEED65
CHECK (US_SL <=65) ;
```



Create Trigger statement

- For example

```
create trigger reorder
  after update
  of qty on stock
  referencing new as n
  for each row mode db2sql
  when (n.qty <=5)
insert into reorder values (n.itemno, current timestamp) ;
```

Manage and query time-based data using temporal tables

- Use temporal tables associated with Time Travel Query to assign time-based state information to your data.
 - Data in tables that do not use temporal support represents the present
 - Data in temporal tables is valid for a period defined by the database system, customer applications, or both
- System-period temporal tables
 - DB2 can automatically store the history of a table
 - The history table contains deleted rows or the original values of rows that have been updated so you can query the past state of your data
- Application-period temporal tables
 - You can also assign a date range to a row of data to indicate when it is deemed to be valid by your application or business rules
- Bi-temporal tables
 - Combine application-period (ATT) and system-period (STT) capabilities

How to Define a System-Period Temporal Table

1. CREATE a table with a SYSTEM_TIME attribute

```
CREATE TABLE travel(  
  trip_name CHAR(30) NOT NULL PRIMARY KEY,  
  destination CHAR(12) NOT NULL,  
  departure_date DATE NOT NULL,  
  price DECIMAL (8,2) NOT NULL,  
  sys_start TIMESTAMP(12) NOT NULL generated always as row begin implicitly  
  hidden,  
  sys_end TIMESTAMP(12) NOT NULL generated always as row end implicitly  
  hidden,  
  tx_start TIMESTAMP(12) generated always as transaction start id implicitly  
  hidden,  
  PERIOD SYSTEM_TIME (sys_start, sys_end)) in travel_space;
```



Captures the begin and end times when the data in a row is current

2. CREATE the history table

```
CREATE TABLE travel_history like travel in hist_space;
```

3. ADD VERSIONING to the system-period temporal table to establish a link to the history table

```
ALTER TABLE travel ADD VERSIONING USE HISTORY TABLE travel_history;
```

Query using a System-Period Temporal Table

- Query the past: what trips were available on 03/01/2012 for less than \$500?
 - Current date = May 1, 2012

```
SELECT trip_name FROM travel FOR SYSTEM_TIME AS OF '03/01/2012'  
WHERE price < 500.00
```

- Query the present: what trips are currently available to Brazil?

```
SELECT trip_name FROM travel WHERE destination = 'Brazil'
```

Defaults to the current table only - functions as if we added
FOR SYSTEM TIME AS OF CURRENT DATE

- Query the past and the present: In 2011, how many different tours were offered?

```
SELECT COUNT (DISTINCT trip_name) FROM travel  
FOR SYSTEM_TIME BETWEEN '01/01/2011' AND '01/01/2012'
```

Example of a Application-period temporal table

- The policy_info table stores the insurance coverage level for a customer
 - The BUSINESS_TIME period-related columns (bus_start and bus_end) indicate when an insurance coverage level is valid
 - A PRIMARY KEY declaration is used when creating the policy_info table, ensuring that overlapping periods of BUSINESS_TIME are not allowed.
 - This means that there cannot be two versions of the same policy that are valid at the same time.

```
CREATE TABLE policy_info
( policy_id CHAR(4) NOT NULL,
  coverage INT NOT NULL,
  bus_start DATE NOT NULL,
  bus_end DATE NOT NULL,
  PERIOD BUSINESS_TIME(bus_start, bus_end),
  PRIMARY KEY(policy_id, BUSINESS_TIME WITHOUT OVERLAPS)
);
```

Using an application-period temporal table

- Query with FOR BUSINESS_TIME AS OF specified

```
SELECT policy_id, coverage, bus_start, bus_end
FROM policy_info
FOR BUSINESS_TIME AS OF '2008-07-15'
where policy_id = 'A123'
```

- Query with FOR BUSINESS_TIME FROM...TO specified

```
SELECT policy_id, coverage, bus_start, bus_end
FROM policy_info
FOR BUSINESS_TIME FROM '2008-01-01' TO '2008-06-15'
where policy_id = 'A123'
```

- The coverage for policy A123 shows an increase from 12000 to 16000 on July 1 (2008-07-01), but an earlier increase to 14000 is missing:

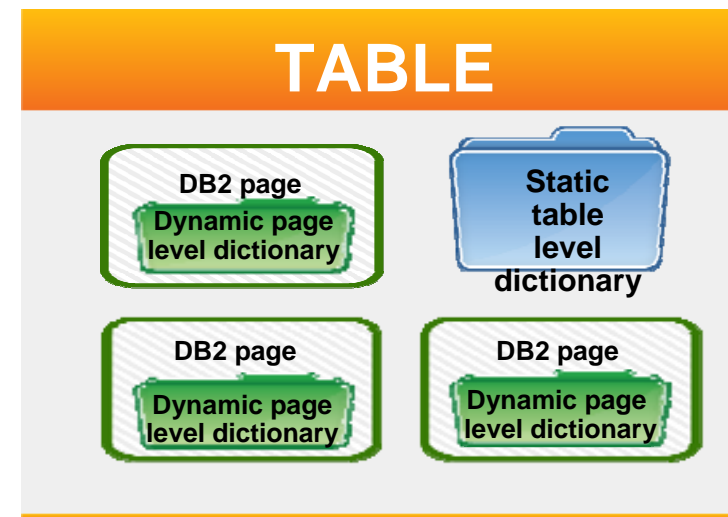
```
UPDATE policy_info
FOR PORTION OF BUSINESS_TIME
FROM '2008-06-01' TO '2008-08-01'
SET coverage = 14000 WHERE policy_id = 'A123';
```


Data Row Compression summary

- Data row compression was introduced in DB2 9.1
- COMPRESS option for CREATE and ALTER TABLE is used to specify compression
- Compression uses a static dictionary:
 - Dictionary stored in data object, about 100K in size
 - Compression Dictionary needs to be built before a row can be compressed
 - Dictionary can be built or rebuilt using REORG TABLE offline, which also compresses existing data
 - A dictionary will be built automatically when a table reaches a threshold size (about 2 MB). This applies to SQL INSERT as well as IMPORT and LOAD (DB2 9.5).
- Compression is intended to:
 - Reduce disk storage requirements
 - Reduce I/Os for scanning tables
 - Reduce buffer pool memory for storing data
- Compression for Indexes, Temporary data and XML data was added in DB2 9.7

Adaptive Compression with DB2 10.1

- Adaptive Compression is an enhancement to the Classic Row Compression found in DB2 9.7
 - Compress rows by using a combination of two types of dictionaries
 - Global static table-level dictionary
 - Local page -level dictionaries
- Benefits
 - Page level dictionaries adapt to data skew over a period of time
 - **No REORGs required** to maintain high compression ratio as data changes
 - Less disk space for data and logs
 - 2x storage savings for tables over Classic Row Compression
 - 5x-8x overall table compression
 - Reduced I/O - Fewer pages to process



- **Better compression ratios than Classic Row Compression**
- **Over time reduces need of a REORG to find locally recurring patterns**



How Does Adaptive Compression Work? Step 1

- **Step 1: Compression with static table level dictionary**

Christine	Haas	(408) 463-1234	555 Bailey Avenue	San Jose	California	95141
John	Thompson	(408) 463-5678	555 Bailey Avenue	San Jose	California	95141
Jose	Fernandez	(408) 463-1357	555 Bailey Avenue	San Jose	California	95141
Margaret	Miller	(408) 463-2468	555 Bailey Avenue	San Jose	California	95141
Bruce	Kwan	(408) 956-9876	4400 North 1st	San Jose	California	95134
James	Geyer	(408) 956-5432	4400 North 1st	San Jose	California	95134
Linda	Hernandez	(408) 956-9753	Street	San Jose	California	95134
Theodore	Mills	(408) 927-8642	650 Harry Road	San Jose	California	95134
Susan	Stern	(408) 927-9630	650 Harry Road	San Jose	California	95134
James	Polaski	(415) 545-1423	425 Market Street	San Francisco	California	94105
John	Miller	(415) 545-5867	425 Market Street	San Francisco	California	94105
James	Walker	(415) 545-4132	425 Market Street	San Francisco	California	94105
Elizabeth	Brown	(415) 545-8576	425 Market Street	San Francisco	California	94105
Sarah	Johnson	(415) 545-1928	425 Market Street	San Francisco	California	94105

California	
[1]	9
[2]	San
[3]	Jose
[4]	Francisco
[5]	Avenue
[6]	Street
[7]	Road

Compression with
global table static
dictionary

- Table-level compression symbol dictionary containing globally recurring patterns
- Table-level dictionary can only be rebuilt during classic table REORG
 - Involves re-compressing all data

Christine	Haas	(408) 463-1234	555 Bailey [5]	[2][3]	[1]	95141
John	Thompson	(408) 463-5678	555 Bailey [5]	[2][3]	[1]	95141
[3]	Fernandez	(408) 463-1357	555 Bailey [5]	[2][3]	[1]	95141
Margaret	Schneider	(408) 463-2468	555 Bailey [5]	[2][3]	[1]	95141
Bruce	Kwan	(408) 956-9876	4400 North 1st [6]	[2][3]	[1]	95134
James	Geyer	(408) 956-5432	4400 North 1st [6]	[2][3]	[1]	95134
Linda	Hernandez	(408) 956-9753	4400 North 1st [6]	[2][3]	[1]	95134
Theodore	Mills	(408) 927-8642	650 Harry [7]	[2][3]	[1]	95134
Susan	Stern	(408) 927-9630	650 Harry [7]	[2][3]	[1]	95134
James	Polaski	(415) 545-1423	425 Market [6]	[2][4]	[1]	94105
John	Miller	(415) 545-5867	425 Market [6]	[2][4]	[1]	94105
James	Walker	(415) 545-4132	425 Market [6]	[2][4]	[1]	94105
Elizabeth	Miller	(415) 545-8576	425 Market [6]	[2][4]	[1]	94105
Sarah	Johnson	(415) 545-1928	425 Market [6]	[2][4]	[1]	94105

How Does Adaptive Compression Work? Step 2

- Step 2: Compression with Page-Level Dictionaries**

Data page

Christine	Haas	(408) 463-1234	555 Bailey [5]	[2][3]	[1]	5141
John	Thompson	(408) 463-5678	555 Bailey [5]	[2][3]	[1]	5141
Ellen	Fernandez	(408) 463-1357	555 Bailey [5]	[2][3]	[1]	5141
Margaret	Schneider	(408) 463-2468	555 Bailey [5]	[2][3]	[1]	5141
Bruce	Kwan	(408) 956-9876	4400 North 1st [6]	[2][3]	[1]	5134
James	Geyer	(408) 956-5432	4400 North 1st [6]	[2][3]	[1]	5134
Linda	Hernandez	(408) 956-9753	4400 North 1st [6]	[2][3]	[1]	5134

Data page

[9]odore	Mills	(408) 927-8642	650 Harry [7]	[2][3]	[1]	5134
Susan	Stern	(408) 927-9630	650 Harry [7]	[2][3]	[1]	5134
James	Polaski	(415) 545-1423	425 Market [6]	[2][4]	[1]	4105
John	Miller	(415) 545-9876	425 Market [6]	[2][4]	[1]	4105
James	Walker	(408) 956-4132	425 Market [6]	[2][4]	[1]	4105
[8]	Miller	(408) 956-8576	425 Market [6]	[2][4]	[1]	4105
Sarah	Johnson	(408) 956-1928	425 Market [6]	[2][4]	[1]	4105

Christine	Haas	(2)1234	(4)
John	Thompson	(2)5678	(4)
Ellen	F(1)	(2)1357	(4)
Margaret	Schneider	(2)2468	(4)
Bruce	Kwan	(3)9876	(5)
James	Geyer	(3)5432	(5)
Linda	H(1)	(3)9753	(5)

[9]odore	(3)s	(4)8642	(6)
Susan	Stern	(4)9630	(6)
(1)	Polaski	(5)1423	(7)
(2)	(3)er	(5)9876	(7)
(1)	Walker	(5)4132	(7)
[8]	(3)er	(5)8576	(7)
Sarah	(2)son	(5)1928	(7)

(1)	ernandez
(2)	(408) 463-
(3)	(408) 956-
(4)	555 Bailey [5] [2][3] [1] 5141
(5)	4400 North 1st [6] [2][3] [1] 5134

Page level dictionary

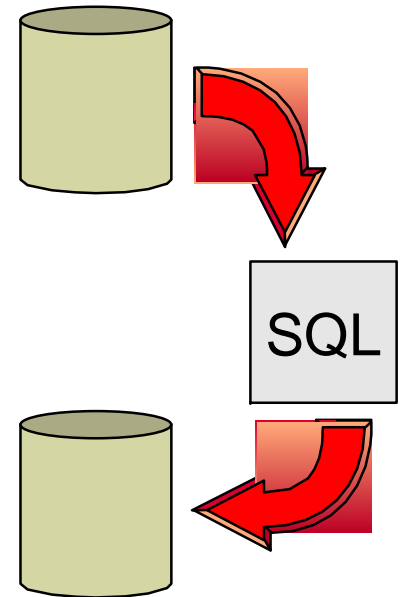
(1)	James
(2)	John
(3)	Mill
(4)	(408) 927-
(5)	(408) 956-
(6)	650 Harry [7] [2][3] [1] 5134
(7)	425 Market [6] [2][4] [1] 4105

Page level dictionary

- Page-level compression dictionaries contain locally frequent patterns
- Page-level compression dictionary building and rebuilding is fully automatic
- Algorithm optimized for compressing data already compressed by table-level dictionary
- Page-level compression dictionaries are stored as special records in each page

db2look utility

- DDL and statistics extraction tool, used to capture table definitions and generate the corresponding DDL.
- In addition to capturing the DDL for a set of tables, can create a test system that mimics a production system by generating the following things:
 - The UPDATE statements used to replicate the statistics on the objects
 - The UPDATE DB CFG and DBM CFG statements for replicating configuration parameter settings
 - The db2set statements for replicating registry settings
 - Statements for creating partition groups, buffer pools, table spaces, and so on



db2look examples

- To capture all of the DDL for a database (includes all tables, views, RI, constraints, triggers, and so on):

```
db2look -d proddb -e -o statements.sql
```

```
{Edit the output file and change the database name}
```

```
db2 -tvf statements.sql
```

- To capture the DDL for one table in particular (table1 in this example):

```
db2look -d proddb -e -t table1 -o statements.sql
```

```
{Edit the output file and change the database name}
```

```
db2 -tvf statements.sql
```

- To capture the DDL for all of the tables belonging to a particular schema (db2user in this example):

```
db2look -d proddb -e -z db2user -o statements.sql
```

```
{Edit the output file and change the database name}
```

```
db2 -tvf statements.sql
```

Unit summary

Having completed this unit, you should be able to:

- Describe the DB2 object hierarchy
- Create the following objects:
 - Schema, Table, View, Alias, Index
- Review the use of Temporary Tables
- Explore the use and implementation of Check Constraints, Referential Integrity and Triggers
- Explain the difference between System-period temporal tables and Application-period temporal tables
- List the types of compression available for tables and indexes
- Use the db2look utility to export database structures for future use

Student exercise

