1. Overview

Informix Dynamic Server Administration
Informix Dynamic Server Terminology
Sysmaster & Sysutils Databases
Informix Dynamic Server Data Types

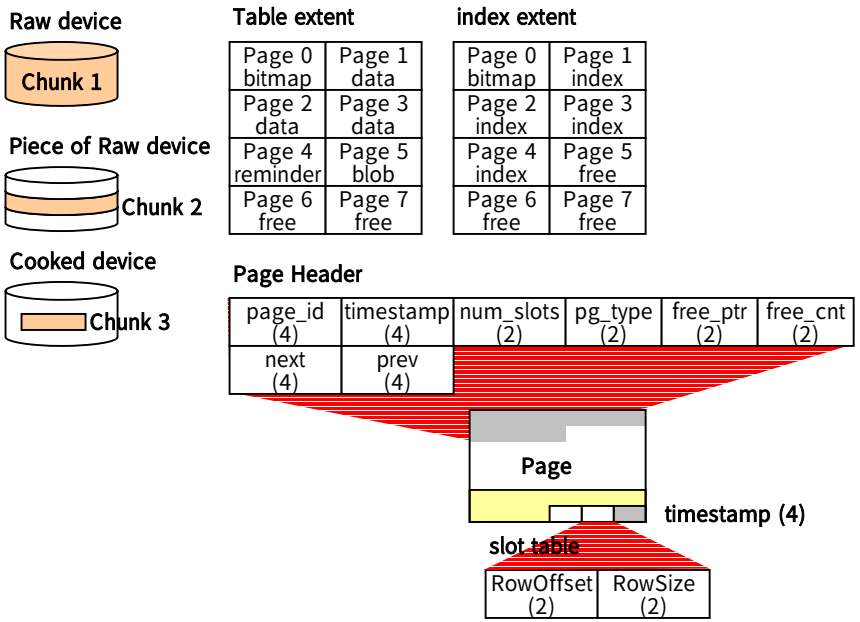
Informix Dynamic Server Administration

- **데이터베이스 및 테이블 생성**
- **데이터베이스 보안 설정**
- **데이터 무결성 (integrity) 보장**
- **동시성 (concurrency) 관리**
- **인덱스 생성**
- **데이터베이스의 최적화**

- 데이터베이스 생성 중 고려사항
  - 데이터베이스의 생성 위치
  - 데이터베이스의 logging 종류
- 테이블 생성 중 고려사항
  - 데이터 형(type)
  - 분할(fragment)
  - 테이블의 생성 위치
  - 디스크 공간 할당 크기
  - lock mode
- 데이터베이스 보안 설정 단위
  - 데이터베이스
  - 테이블
  - 뷰(view)
  - 컬럼
  - 형(type)
  - 분할 (fragment)
  - 루틴(stored procedure 또는 Function)
  - 언어(SPL, C, JAVA)
- 데이터 무결성 (Integrity)
  - Referential Constraints : 테이블 사이의 관계
  - Entity Constraints : unique한 primary key
  - Semantic Integrity Constraints : 데이터 형, default값, check
- 동시성 (concurrency) 관리
  - Read Concurrency : isolation level로 구현
  - Update Concurrency : Insert, update, delete 문 사용 중 lock 관리
- 최적화
  - 인덱스 생성
  - Update statistics
  - Data Distributions
  - SQL Statement Cache
  - Memory Resident Tables
  - Triggers and Routines

Informix Dynamic Server Terminology

- **Chunk**
- **Page**
- **Extent**
- **Tblspace**
- **Dbspace**
- **Blobspace**
- **Smart blobspace**



- **Chunk**
  - 서버에 연속적으로 할당된 물리적 공간의 단위로, 절대경로명과 offset에 의해 식별할 수 있다.
  - chunk는 mount되지 않은 character special device인 raw disk와 offset에 의해 지정된 raw disk의 일부분, mount된 파일시스템 영역의 파일로 사용할 수 있다.
- **Page**
  - 저장 및 I/O의 기본 단위로 크기는 특정 하드웨어 및 O/S에 의해 결정되어 있으므로 변경할 수 없다. 대부분의 시스템이 일반적으로 2K 크기 인데, 일부는 4K 페이지를 사용하기도 한다.한 page에는 특정 유형(데이터, 인덱스, 비트맵 등)의 데이터가 저장된다.
- **Extent**
  - 물리적으로 연속적인 페이지들의 모임으로, 테이블에 디스크 영역을 할당하는 단위이다. extent의 크기는 테이블을 만들 때 지정한다.
    - EXTENT SIZE : 초기크기
    - NEXT SIZE : 추가할 때 사용되는 크기
  - extent의 최소크기는 4 페이지이고 기본값은 8 페이지이다.
- **Tblspace**
  - 한 테이블을 구성하는 모든 데이터와 인덱스 페이지를 논리적으로 구분하는 단위로 반드시 연속적이라 볼 수 없다.
- **Dbspace**
  - chunk의 논리적인 묶음. 인포믹스 서버는 적어도 하나의 DBspace가 필요한데, 이것을 root dbspace라 한다.
  - 임시테이블 또는 임시 파일을 저장하는 특수한 영역을 temp dbspace라 한다.
- **Blobspace**
  - TEXT, BYTE 데이터를 저장하는 특수한 dbspace. blobspace를 생성하면서 정의한 blob page는 연속된 페이지의 묶음으로 구성된다
- **Sbspace**
  - Smart Large Object (BLOB, CLOB), User-defined Data type을 저장하는 특수한 dbspace

Sysmaster & Sysutils Databases

- **sysmaster**
  - **System Monitoring Interface (SMI)**
  - **서버 인스턴스의 상태 및 진단 정보를 수집하는데 사용**
  - **공유 메모리의 내용을 실시간으로 접근**
- **sysutils**
  - **ISM(Informix Storage Manager) 와 onbar 유틸리티에서 사용**

※ system catalog table은 데이터베이스가 생성될 때 해당 데이터베이스에 자동적으로 생성된다

- Sysmaster 데이터베이스는 서버가 처음으로 on-line상태가 될 때 (초기화과정) 자동적으로 생성된다
- Sysmaster 데이터베이스의 테이블 대부분이 view형태로 메모리 구조를 그대로 pointing하고 있으므로 SQL문장을 통하여 메모리 상태를 실시간으로 access할 수 있다.

```
Informix> vi example.sql
SELECT dbsnum, name, owner, nchunks, is_temp, is_blobspace, flags
  FROM sysdbspaces;

SELECT c.dbsnum, chknum, nxchknum, d.name, chksize, nfree
  FROM syschunks c, sysdbspaces d
 WHERE c.dbsnum = d.dbsnum
   AND d.name = 'dbspace_name';

SELECT name, owner, created, is_logging, is_buff_log, is_ansi
  FROM sysdatabases;

SELECT partnum, dbsname, owner, tabname
  FROM systabnames
 WHERE dbsname = 'database_name';

SELECT sid, username, uid, hostname
  FROM syssessions;

Informix> dbaccess sysmaster example.sql
```

서버의 모든 dbspace정보
– chunk수, dbspace종류, 상태

특정 dbspace의 chunk정보
– free page수, 사용된 page수

서버의 모든 데이터베이스와 logging 상태

특정 데이터베이스의
모든 테이블 이름과 생성한 사용자명

세션정보
– 세션번호, 사용자명, 호스트서버

- sysutils 데이터베이스도 sysmaster 데이터베이스와 마찬가지로 서버가 초기화 되어 처음으로 on-line상태가 될 때 자동적으로 생성된다
  - bar_object
  - bar_action
  - bar_version
  - bar_object

- 실습 및 테스트에 필요한 데모 데이터베이스는 다음과 같이 생성한다.
  - dbaccessdemo –log database_name

Informix Dynamic Server Data Types

- **Built-in Data Types**
  - **Character**
  - **Boolean**
  - **Numeric**
  - **Time**
  - **Large-Object**
- **Extended Data Types**
  - **Complex Data Types**
  - **User-Defined Data Types**

- **Character**
  - CHAR (n)
  - VARCHAR (m,r)
  - LVARCHAR
- **BOOLEAN**
- **Numeric**
  - SMALLINT
  - INTEGER / INT8
  - FLOAT
  - SMALLFLOAT
  - DECIMAL (p,s)
  - MONEY (p,s)
  - SERIAL / SERIAL8
- **Time**
  - DATE
  - DATETIME
  - INTERVAL
- **Large-Objects**
  - Simple Large-Objects
    - TEXT
    - BYTE
  - Smart Large-Objects
    - BLOB
    - CLOB
- **Complex Data Types**
  - Collection Data Types
    - SET
    - MULTISET
    - LIST
  - Row Data Types
    - Named row types
    - Unnamed row types
- **User-Defined Data Types**
  - Distinct Types
  - Opaque Types

Informix Dynamic Server Data Types – Character, Boolean, Numeric types

- **CHAR(n)**
- **VARCHAR(m, r)**
- **LVARCHAR**
- **BOOLEAN**
- **SMALLINT**
- **INTEGER**
- **INT8**
- **FLOAT**
- **SMALLFLOAT**
- **DECIMAL (p,s)**
- **MONEY (p,s)**
- **SERIAL (n)**
- **SERIAL8 (n)**

Type	최소	최대	비고
smallint	$-(2^{15}-1)$: -32,767	$2^{15}-1$: 32,767	2 byte
integer	$-(2^{31}-1)$: -2,147,483,647	$2^{31}-1$: 2,147,483,647	4 byte
int8	$-(2^{63}-1)$	$2^{63}-1$	8 byte (64bit) 10 byte (32bit)
smallfloat	최대 8자리 유효자리수		4 byte
float	최대 16자리 유효자리수		8 byte
decimal money	precision, scale을 지정 최대 32자리 유효자리수 money는 화폐기호 ($DBMONEY) 출력 기본값 : decimal(16,0), money(16,2)		scale이 홀수:(p+4)/2 byte 짝수:(p+3)/2 byte

- **Character Data Type**
  - CHAR(n)
    - 최대 32,767 byte, 최소 1byte의 문자를 저장
    - 고정길이(Fixed Length)로 저장되므로 실제 데이터의 크기가 작더라도 정의된 크기 만큼 공간 할당
    - char type에 저장된 숫자에 sum,avg같은 함수는 사용 불가능
  - VARCHAR(m,r)
    - 최대크기(max)와 예약크기(reserved)를 지정. 예약크기는 선택
    - 예약 크기 만큼만 공간 할당, 최대 크기까지 증가 가능
    - 최대크기는 255byte
    - 내부적으로 1byte의 indicator가 추가로 저장
  - LVARCHAR
    - 최대 2Kbytes 의 문자 저장
    - 내부적으로 3byte의 indicator가 추가로 저장
- **BOOLEAN**
  - 1byte의 참(true)과 거짓(false) 값을 저장
  - 대소문자 구분 없음 ('t'='T', 'f'='F')
- **Numeric Data type**
  - 왼쪽 표 참조
  - Decimal은 float와 달리 truncate되지 않고 round 된다
  - Decimal(5,2)의 sample 데이터 : 123.45
  - float 의 유효자리는 하드웨어에 따라 다를 수 있다
  - SERIAL 및 SERIAL8 은 각각 INTEGER 와 INT8 로 저장되는 type인데, 데이터가 입력될 때 자동적으로 1씩 증가시킬 수 있는 특징이 있다. 시작값 n을 지정하지 않으면 1부터 시작되고, 행이 삭제되더라도 SERIAL 값은 계속 증가한다.
  - 한 테이블에 SERIAL 컬럼 하나만 선언할 수 있다.
  - SERIAL 및 SERIAL8 컬럼에 수동으로 특정 값을 입력할 수 있으므로 항상 unique하다고 보장할 수 없으므로 unique한 성질을 유지시키려면 unique index나 primary key를 이 컬럼에 생성해야 한다.

Informix Dynamic Server Data Types – Time types

- **DATE**
- **DATETIME**
- **INTERVAL**

※ 관련 환경 변수
DBDATE
DBCENTURY

입력값		99.06.04	02.06.04	02.12.31
DBCENTURY 설정에 따라 저장된 값 ※ 오늘날짜 : 2002.06.04	P	1999.06.04	1902.06.04	1902.12.31
	F	2099.06.04	2102.06.04	2002.12.31
	C	1999.06.04	2002.06.04	2002.12.31
	R	2099.06.04	2002.06.04	2002.12.31

P : past
F : Future
C : Closest
R : Present (Default)

- **DATE**는 integer로 저장되는 날짜로 1900년 1월 1일을 기준값 1로 시작하여 1씩 증가한다.
- DATE가 날짜 형태로 표현되는 포맷은 환경변수 DBDATE에 의해 결정된다.
- 기본 DBDATE는 mdy4/ 이다. (mm/dd/yyyy)
  - DBDATE=y4md- : yyyy-mm-dd
  - DBDATE=y2md : yy/mm/dd
  - DBDATE=y4md0 : yyyymmdd
  - DBDATE=mdy2. : mm.dd.yy
- 두자리 연도로 입력되는 날짜의 경우 환경변수 DBCENTURY에 의해 정확한 연도가 결정된다. (왼쪽 표 참조)
  - DBCENTURY=P : (past) 현재 날짜 보다 모두 과거
  - DBCENTURY=F : (future) 현재 날짜 보다 모두 미래
  - DBCENTURY=C : (closet) 현재 날짜에 가까운 세기의 연도
  - DBCENTURY=R : (present) 현재 세기, 기본값
- **DATETIME** 은 년도부터 초단위 이하 시간까지 지정한 범위의 날짜 및 시간값을 저장한다
  - DATETIME(2002-05-31 19:00) YEAR TO MINUTE
  - DATETIME(17 04:35:48) DAY TO SECOND
  - DATETIME(03-25 12:04:25.085) MONTH TO FRACTION
- DATETIME은 다음과 같은 범위에서 사용할 수 있다
  - YEAR : 1~9999
  - MONTH : 1 ~ 12
  - DAY : 1 ~ 28,29,30,31
  - HOUR : 0 ~ 23
  - MINUTE : 0 ~ 59
  - SECOND : 0 ~ 59
  - FRACTION(n) : n의 기본값은 3, 1~5까지 가능. 현재 날짜와 시간을 표시하는 current함수를 사용할 때 소수점 이하 n자리 까지 정확한 데이터가 입출력 되려면 $ONCONFIG의 USEOSTIME이 1로 설정되어야 한다
- **INTERVAL**은 시간차를 나타내는데, 다음 두가지 분류로 나누어진다
  - Year-month interval class : YEAR, MONTH
  - Day-time interval class : DAY,HOUR,MINUTE,SECOND,FRACTION
  - INTERVAL(5-3) YEAR TO MONTH
  - INTERVAL(15:26) HOUR TO MINUTE

Informix Dynamic Server Data Types – Large Objects, Collection types

- **TEXT**
- **BYTE**
- **CLOB**
- **BLOB**
- **SET**
- **MULTISET**
- **LIST**
- **ROW**

- **Simple Large Objects**
  - 일반 dbspace나 blobspace에 저장
  - 한 행, 한 컬럼에 최대 2G 까지 저장 가능
  - 테이블의 데이터 행에는 large object의 위치를 가리키는 56byte의 정보만 저장되고 large object는 데이터 행과 다른 별도의 공간에 저장된다
  - random access는 지원하지 않지만 일부 영역 검색은 가능하다
    select cat_descr[1,20] from catalog where catalog_num = 10001
  - ASCII 데이터를 저장하는 TEXT type과 Binary 데이터를 저장하는 BYTE type을 제공한다
- **Smart Large Objects**
  - 항상 sbspace에 저장
  - 최대 4TB 까지 저장 가능
  - large object에 대해 logging이 가능
  - random access 지원
  - ASCII 데이터를 저장하는 CLOB type과 Binary 데이터를 저장하는 BLOB type을 제공
- sbspace는 다음과 같은 onspaces 명령으로 생성한다
  - onspaces –c –S sbspace_name –p chunk_path –o offset –s size

```
Informix> vi slob.sql
CREATE TABLE slob_tab ( c_file clob, b_file blob )
   PUT c_file in (sbspace), b_file in (sbspace);
INSERT INTO slob_tab VALUES (filetoclob("/etc/passwd","server"),
                             filetoblob("/bin/ls","server"));

Informix> vi set.sql
CREATE TABLE flag_color (country CHAR(30), color SET(char(20) NOT NULL));
INSERT INTO flag_color VALUES ("USA",SET{"red","white","blue"});

Informix> vi row.sql
create row type my_row (a int, b int);
create table named_row (a int, b my_row);
insert into named_row values (1,row(2,3)::my_row);
create table unnamed_row (a int,b row(a int,b int));
insert into unnamed_row values (1,row(2,3));

Informix> vi distinct.sql
create distinct type inch as decimal;
create table part (length inch,width inch,height inch);
insert into part values (1.0::inch,2.0::inch,3.0::inch);
```

- **Collection type**
  - SET : unordered, unique
  - MULTISET : unordered, non-unique
  - LIST : ordered, non-unique
- **Row data type**
  - Named : row type을 먼저 생성
  - Unnamed : 테이블 정의 할 때 row 키워드로 직접 생성
- **Distinct type**
  - 기존 데이터 type을 이용하여 생성한 새로운 이름의 type
- **Opaque type**
  - C, Java같은 언어로 생성된 기존에 존재하지 않는 새로운 type

2. Database Physical Modeling

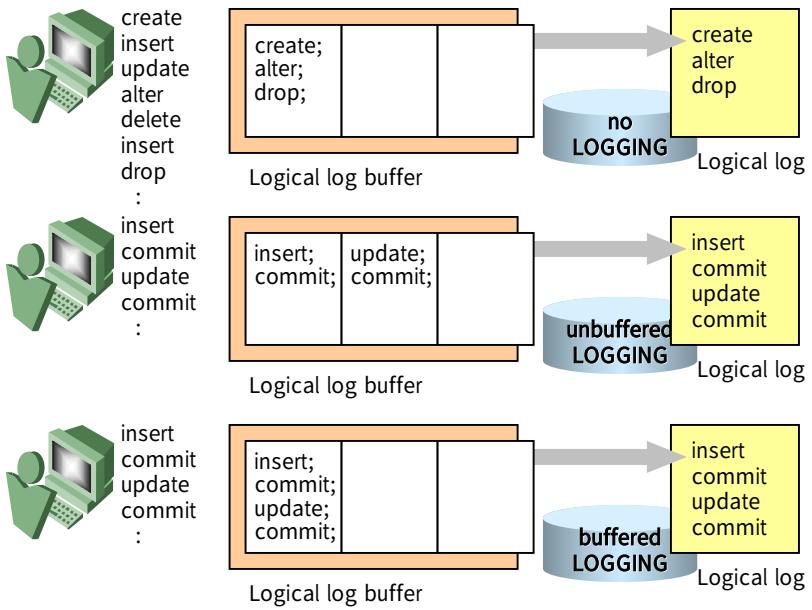
Creating a Database
Creating a Table
Altering a Table
Renaming / Dropping Tables and Databases
Performance Tuning for Tables
Table / Database Monitoring
Index Structure
Creating / Altering / Dropping Indexes
Indexing Guidelines
Performance Tuning for Indexes
Table Partitioning Overview
Fragmenting a Table / Index
Fragment Guideline
ROWIDS / Data Skip etc.

Creating a Database

- **데이터베이스 생성 시 고려 사항**
  - **데이터베이스 이름**
  - **Logging Mode 결정**
    - **No Logging**
    - **Buffered Logging**
    - **Unbuffered Logging**
    - **Mode-ANSI**
  - **생성될 영역 (dbspace)**

※ Mode-ANSI 를 제외한 모든 logging 모드의 데이터베이스에서 테이블을 생성하면 자동적으로 public에게 select,insert,delete,update,index 사용권한이 부여된다. 이러한 기능을 막고자 한다면 테이블을 생성하기 전에 다음 환경변수를 설정한다

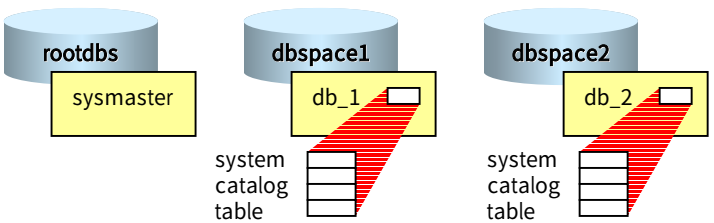
export NODEFDAC=yes



- 데이터베이스 이름 규칙
  - 최대 128자
  - A~Z,a~z,0~9,_ 로 구성
  - 하이픈(-),공백 문자나 알파벳이 아닌 문자는 사용 불가능
  - 데이터베이스 서버 안에서 유일
- 트랜잭션을 사용할 때 트랜잭션 정보가 메모리의 logical log buffer에 기록되고 이 내용이 디스크에 존재하는 logical log에 flush되는 과정을 데이터베이스 logging이라 한다
- **No Logging**
  - Update, Delete, Insert와 같은 DML문은 logical log에 기록되지않고 Create, Alter와 같은 DDL문 및 관리작업 내용은 logical log에 기록된다.
  - 대량의 테이블을 로드 할 때는 No Logging 상태에서 작업을 해야 속도가 빠르고 long transaction을 피할 수 있다.
  - DML문을 기록하지 않으므로 트랜잭션 정보를 잃어버리게 되고, 백업으로부터 완벽한 복구가 불가능 하다.
  - 다음 명령으로 데이터베이스를 no logging 모드로 바꿀 수 있다
    - ontape –N <database_name>
- **Unbuffered Logging**
  - 모든 SQL문이 logical log buffer에 기록되고 commit될 때 logical log로 flush된다
  - commit된 트랜잭션은 디스크에 기록이 되어 있는 상태이므로 트랜잭션의 손실 없이 완벽한 복구가 가능하다.
- **Buffered Logging**
  - logical log buffer가 가득찰 때까지 모든 SQL문이 buffer에 기록된다.
  - 디스크로 flush하는 회수가 unbuffered log에 비하여 적으므로 성능은 좋으나 flush하기 전에 시스템 장애가 발생하면 메모리에 있는 트랜잭션 정보는 잃게 된다
- **Mode-ANSI**
  - Unbuffered Logging과 같이 logical log buffer를 flush하는데, 모든 SQL문이 자동으로 transaction이 시작되는 implicit transaction을 사용하므로 BEGIN WORK문장이 필요 없고 COMMIT WOK 또는 ROLLBACK WORK 문장만 사용한다.
  - logging 모드를 변경할 수 없고 기본 isolation은 repeatable read이다 (isolation은 “3.Data Control”에서 설명된다)
  - 테이블에는 항상 owner이름을 같이 써야 한다.
  - 테이블을 생성할 때 자동적으로 public에게 권한이 부여되지 않는다

Creating a Database

- **db_dbs 영역에 no-logging으로 데이터베이스 생성**
  CREATE DATABASE db IN db_dbs
- **db_dbs 영역에 unbuffered logging으로 데이터베이스 생성**
  CREATE DATABASE db IN db_dbs WITH LOG
- **db_dbs 영역에 buffered logging으로 데이터베이스 생성**
  CREATE DATABASE db IN db_dbs WITH BUFFERED LOG
- **시스템 기본 영역(rootdbs)에 Mode ANSI로 데이터베이스 생성**
  CREATE DATABASE db WITH LOG MOD ANSI



- 데이터베이스를 생성하면 자동적으로 그 데이터베이스 안에 system catalog 테이블이 생성된다. System catalog 테이블은 사용자가 그 데이터베이스에서 생성한 모든 테이블, 인덱스 등 모든 데이터 오브젝트를 관리한다.
- 데이터베이스를 생성할 때 특정 영역을 지정할 수 있는데, 이 영역에 system catalog 테이블이 생성된다. 또한 이 데이터베이스 안에서 생성하는 테이블의 기본 위치가 된다.
- 데이터베이스의 생성 영역을 지정하지 않으면 기본 위치로 root dbspace에 데이터베이스가 생성된다. root dbspace는 계속 변경이 일어나는 sysmaster 데이터베이스가 생성된 영역이기 때문에 I/O가 빈번하므로 데이터베이스를 생성할 때는 항상 기본 영역 외의 다른 영역을 지정해야 한다.

Creating a Table

- **테이블 생성 시 고려사항**
  - **테이블 이름**
  - **적절한 컬럼과 자료형**
  - **Extent 크기**
  - **Lock 모드**
  - **저장될 공간 (dbspace)**
  - **테이블의 logging 여부**
- **Extent**
  - **테이블을 생성할 때 물리적으로 연속되게 할당되는 공간의 단위**
  - **최소 4 페이지**
  - **EXTENT SIZE : 처음 할당될 extent 크기 (기본값 : 8페이지)**
  - **NEXT SIZE : 추가될 extent 크기 (기본값 : 8페이지)**

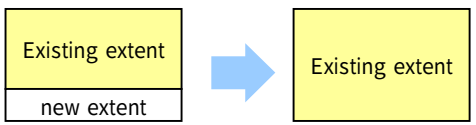
• 테이블 이름 규칙
  – 최대 128 문자
  – 가급적 “$”나 “#” 은 사용하지 않음
  – 알파벳 문자, 숫자, “_” 사용 가능하지만 숫자로 시작하지 않음
  – 대소문자 구분 없음

• DELIMIDENT 환경변수를 설정하게 되면 다음과 같은 경우의 테이블 이름을 사용할 수 있게 된다.
  – “$”, “#”같이 알파벳이 아닌 문자
  – 공백
  – 대소문자 구분
  – 숫자로 시작하는 테이블

```
Informix> export DELIMIDENT=ON
Informix> dbaccess stores_demo –
create table “sharp#sharp” (col1 int);
create table “SHARP#sharp” (col1 int);
create table “S P A C E” (col1 int);
create table “123number” (col1 int);
```

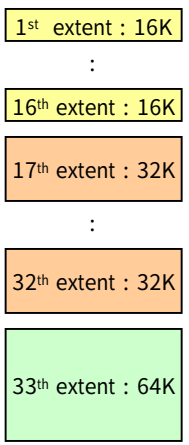
• extents는 다음과 같이 증가 할 수 있다

Concatenation



Manual
Alter table table_name modify next size 128

Doubling



• Extent 크기 결정
  – 1. num_rows = 저장할 데이터의 row 수
  – 2. rowsize = 실제 row의 크기 + 4 bytes
    ※ varchar나 blob 컬럼이 포함된 row는 계산이 정확하지 않음
  – 3. 한 페이지에 사용 가능한 크기
    pageuse = 페이지크기 – 28 bytes
  – 4. 저장될 페이지의 수
    max_rows = min(pageuse/rowsize, 255)
    num_pages = num_rows / maxrows
  – 5. 전체 크기를 kilobyte로 환산
    (num_pages * 페이지크기) / 1024

• extents할당에 대한 정보는 sysmaster 데이터베이스에서 조회 가능

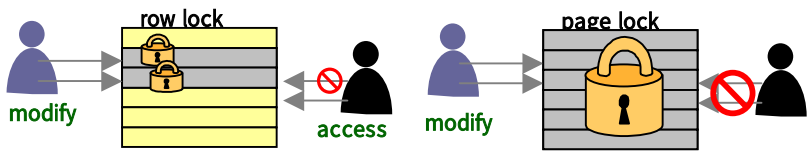
```
Informix> dbaccess sysmaster –
select * from sysextents;
```

Creating a Table

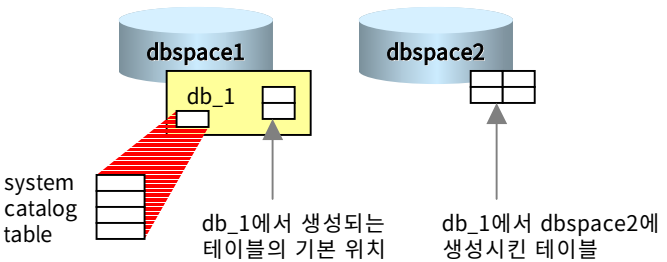
- **테이블 lock 모드**
  - **page : 한 페이지에 저장된 모든 데이터에 대해 locking (default)**
  - **row : 해당되는 row에 대해서 locking**
- **테이블 생성 위치(dbspace) 지정**
  - **테이블과 데이터베이스의 I/O 부하 조정**
  - **테이블이 생성될 공간이 부족할 때**
  - **백업 및 복구 단위 조절**

- Page 레벨 lock은 인접한 행에 대한 lock이 필요할 때 해당 페이지의 데이터를 모두 한번에 locking하기 때문에 자원 활용면에서는 바람직하지만 (lock의 크기는 약 44bytes) concurrency 가 떨어진다는 단점이 있다.
- 테이블을 생성할 때 default lock 모드는 page 레벨인데, 다음과 같은 방법으로 default lock 모드를 변경할 수 있다. (9.3이후)
  - configuration parameter : DEF_TABLE_LOCKMODE row
  - 환경변수 : export IFX_DEF_TABLE_LOCKMODE=row

- Row/Page lock의 concurrency



- 테이블과 데이터베이스의 dbspace



Creating a Table

테이블 생성 구문

```
CREATE TABLE orders
(
   order_num      SERIAL NOT NULL,
   customer_num   INTEGER,
   order_date     DATE
)
IN dbspace2
EXTENT SIZE 64
NEXT SIZE 32
LOCK MODE ROW;
```

컬럼 및 자료형

위치

extent 크기 (kbytes)

lock 모드

- Simple large object 컬럼을 포함한 테이블

```
CREATE TABLE evaluation (
    employee_num  SERIAL,
    manager_num   INTEGER,
    emp_eval_form TEXT IN blobspace1,
    emp_picture   BYTE IN blobspace2,
    emp_phrase    TEXT IN TABLE
)
IN dbspace2;
```

- Smart large object 컬럼을 포함한 테이블

```
CREATE TABLE movie(
    movie_num   INTEGER,
    movie_title CHAR(50),
    video       BLOB,
    audio       BLOB,
    description CLOB
)
PUT video       IN (sbsp3),
    audio       IN (sbsp1),
    description IN (sbsp5);
```

- Temporary 테이블 생성

```
CREATE TEMP TABLE temp_order (order_num INTEGER) WITH NO LOG;
SELECT customer_num, company FROM customer INTO TEMP cust_temp WITH NO LOG;
```

  - DBSPACETEMP 환경변수나 $ONCONFIG 파라매터에 설정된 temp dbspace에 임시 테이블 생성
  - Temp dbspace에서는 테이블에 logging을 하지 않으므로 logging된 데이터베이스에서 임시 테이블을 만들 때 WITH NO LOG옵션을 사용해서 temp dbspace에 테이블이 생성되도록 한다.

- Logging하지 않는 테이블 생성(raw table)
  - 단, index나 referential constraint를 설정할 수 없다

```
CREATE RAW TABLE nolog_table (
    order_num  INTEGER,
    order_date DATE
);
ALTER TABLE nolog_table type (STANDARD);
ALTER TABLE nolog_table type (RAW);
```

Altering a Table

- **Fast ALTER**
  - **Lock 모드 변경**
  - **Next extent 크기 변경**
  - **Constraint 생성 및 삭제**
- **In-place ALTER**
  - **컬럼 추가 및 삭제**
  - **컬럼의 자료형 변경**
  - **Fragment expression변경**
- **Slow ALTER**
  - **Fast ALTER나 In-Place ALTER에 해당하지 않는 모든 경우**

• Fast alter 는 테이블에 저장된 데이터에 영향을 미치지 않고 테이블의 정보가 저장된 system catalog table 만 update 되므로 alter 속도가 빠르다

```
ALTER TABLE orders LOCK MODE (ROW);
ALTER TABLE customer NEXT SIZE 20;
```

  – lock mode는 row 또는 page로 변경할 수 있다
  – 테이블이 생성될 때 이미 초기 extent가 할당되었으므로 next extent만 altering할 수 있다

• 대부분의 alter구문은 저장된 데이터에 직접적으로 영향을 미치는데, 기존에 저장된 데이터에 변경이 일어날 때 처리 시간이 길어지고 데이터에 lock이 걸려 alter가 끝날 때까지 다른 사용자의 작업이 불가능하다.
• In-place alter는 alter를 빠르게 수행하는 알고리즘인데, alter 전,후의 데이터 구조를 버전 관리하여 기존 데이터 구조는 유지한 채 변경되는 데이터 구조를 생성한다. 다음과 같은 경우 in-place alter가 적용된다.
  – 컬럼 추가 및 삭제
    (단, fragment에서 WITH ROWID구문을 사용했거나, replication에서 WITH CRCOLS를 사용했을 경우 제외, 또는 BYTE, TEXT컬럼 삭제의 경우 제외)
  – 컬럼 자료형 변경
  – Fragment의 expression변경
    (단, 데이터가 다른 dbspace로 이동하지 않을 때)

```
ALTER TABLE customer ADD birthday DATE;
ALTER TABLE customer MODIFY birthday DATETIME YEAR TO MINUTE;
ALTER TABLE customer DROP birthday;
```

• Slow alter는 fast alter나 in-place alter 알고리즘을 사용하지 않는 경우 실행된다.
  – alter구문이 처리되는 동안 테이블에는 exclusive 모드의 lock이 걸린다
  – alter가 완료될 때까지 테이블의 새 copy 영역이 생성되고 기존의 데이터가 새 영역으로 이동되므로 충분한 공간이 필요하다.
  – alter에 의해 데이터가 옮겨질 때 모든 데이터 레코드에 대하여 logging을 하기 때문에 long transaction이 발생할 수도 있다.

• Simple LOB를 Smart LOB로 변환 (TEXT를 CLOB으로, BYTE를 BLOB으로)

```
ALTER TABLE booklist MODIFY content CLOB, PUT content IN (sbsp1) (log);
```

Renaming / Dropping Tables and Databases

- **컬럼 이름 변경**
  RENAME COLUMN table_name.old_column_name
  TO table_name.new_column_name;
- **테이블 이름 변경**
  RENAME TABLE old_table_name TO new_table_name;
- **데이터베이스 이름 변경**
  RENAME DATABASE old_db_name TO new_db_name;
- **테이블 삭제**
  DROP TABLE table_name;
- **데이터베이스 삭제**
  DROP DATABASE db_name;

- 컬럼, 테이블, 데이터베이스 이름을 RENAME~TO 구문으로 변경할 수 있는데, 이때 관련된 view와 트리거의 이벤트도 함께 변경된다.
- 하지만, 사용자 정의 루틴 소스나 트리거 액션의 컬럼, 테이블 이름은 변경되지 않는다.

```
Informix> dbaccess stores_demo << !
create table org_1 ( col1 int );
create table org_2 ( col1 int );
create view view_1 as select col1 from org_1;
create trigger trig_1 update of col1 on org_1 referencing new as new
    for each row (insert into org_2 values (new.col1));
rename column org_1.col1 to new_col1;
rename table org_1 to new_1;
!

Informix> dbschema –d stores_demo
create table "informix".new_1
  (
    new_col1 integer
  );
create table "informix".new_2
  (
    col1 integer
  );
create view "informix".view_1 (col1) as
  select x0.new_col1 from "informix".new_1 x0 ;
create trigger "informix".trig_1
    update of new_col1 on "informix".new_1 referencing new as new
    for each row (insert into "informix".org_2 (col1)  values (new.new_col1));
```

변경

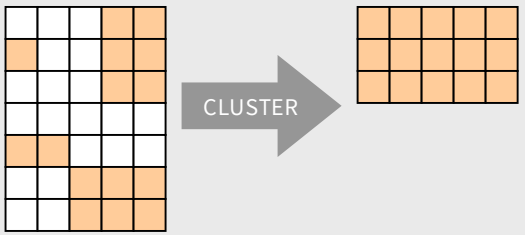
변경되지 않음

- 테이블을 drop하면 테이블과 관련된 뷰나 트리거도 삭제된다.
- 테이블이나 데이터베이스를 삭제하면 사용하던 공간이 free된다.
- 데이터베이스는 삭제 후 rollback할 수 없으므로 주의해야 한다. 만약 데이터베이스를 복구하기 위해서는 백업으로부터 복구해야 한다.

Performance Tuning for Tables

- **테이블의 데이터 메모리 상주 (Memory Resident)**
  SET TABLE table_name MEMORY_RESIDENT;
  SET TABLE table_name NON_RESIDENT;

- **사용하지 않는 영역 반환**
  ALTER INDEX index_name TO CLUSTER;



- Memory resident는 테이블의 데이터를 우선적으로 공유메모리 버퍼 풀에 상주시킨다. 이 데이터를 가장 최근에 메모리로 로드된 것처럼 인식 시키므로 일반 데이터 페이지에 비해 오래 동안 메모리에 상주할 수 있게 된다.

```
dbaccess sysmaster << !
select hex(partnum),partnum
from systabnames
where dbsname = 'stores_demo'
and tabname = 'customer'
!
(expression)     partnum
0x00100270       1049200
```

```
Informix> dbaccess stores_demo << !
set table customer memory_resident;
!
Informix> onstat –T
Tblspaces
 n address  flgs ucnt tblnum physaddr npages nused  npdata nrows  nextns resident
 1 b342018  0    1    100001 10000e   650    650    0      0      9      0
 2 b346d18  0    0    100002 10000f   4      4      2      12     1      0
                                      :
23 b2db690  2000 0    100270 10600a   8      6      3      29     1      2000
                                      :
Informix> dbaccess stores_demo << !
select * from customer;
!
Informix> onstat –P
partnum  total    btree    data     other    resident dirty
0        4861     0        0        4861     0        0
1048577  2        0        0        2        0        0
1048578  5        1        2        2        0        0
                           :
1049200  7        0        5        2        5        0
```

resident flag

resident된 페이지 수

- 테이블에 할당된 extent는 사용을 하지 않더라도 자동적으로 반환하지 않는다. ALTER INDEX ~ TO CLUSTER 구문을 실행하면 인덱스의 데이터 순서로 데이터의 위치를 재배열하는 새 테이블 영역을 만들게 된다.
  – 사용하지 않는 영역 반환
  – extent의 수 감소
- 테이블 당 extent의 최대 개수는 대략 2KB 페이지를 사용하는 시스템에서는 200개, 4KB 페이지를 사용하는 시스템에서는 450개 정도이다.

Table / Database Monitoring

- **dbschema 유틸리티**
  dbschema –d db_name [–t table_name] [–ss] [file_name]

- **oncheck 유틸리티**
  - **테이블의 정보**
  oncheck –pT db_name:table_name

- **onstat 유틸리티**
  - **메모리 상주 테이블 조회**
  onstat –t (active상태) / onstat –T (모든 테이블)
  onstat –P (메모리 상주 페이지 수)

- **sysmaster 및 system catalog 조회**

• dbschema 유틸리티
  – 데이터베이스 전체 또는 특정 테이블의 schema를 알 수 있다
  – "-ss" 옵션을 추가하면 테이블의 dbspace위치, extent 크기, lock mode 등을 알 수 있다.
  – 출력결과를 지정한 파일에 저장할 수 있다

```
Informix> dbschema –d stores_demo –t new_2
create table "informix".new_2
  (
    col1 integer
  );
Informix> dbschema -d stores_demo -t new_2 -ss
create table "informix".new_2
  (
    col1 integer
  ) extent size 16 next size 16 lock mode page;
Informix> dbschema –d stores_demo stores_demo.sql
```

stores_demo.sql 파일로 저장

• 현재 데이터베이스의 테이블 목록 조회

```
SELECT tabname, decode(tabtype,'T','table','V','view','S','synonym',tabtype)
FROM systables WHERE tabid > 99
```

• 현재 시스템의 데이터베이스 목록 및 정보 조회

```
SELECT d.name, d.owner, d.created,
       case (is_logging+(is_buff_log*2)+(is_ansi*4))
       when 0 then 'No_Logging'
       when 1 then 'Unbuffered'
       when 3 then 'Buffered'
       when 5 then 'Ansi'
       end logging , s.name
FROM sysdatabases d, sysdbspaces s WHERE partdbsnum(d.partnum) = s.dbsnum ;
```

Exercise

- **테이블 생성**
  CREATE TABLE monitoring (col1 SERIAL, col2 CHAR(2000));
  CREATE TABLE dummy (col1 INT);
- **데이터 입력 (8건)**
  INSERT INTO monitoring VALUES (0,'data');
- **컬럼 추가**
  ALTER TABLE monitoring ADD col3 INT;
- **데이터 입력 (1건)**
  INSERT INTO monitoring VALUES (0,'data');
- **Monitor**
  oncheck –pT stores_demo:monitoring

• oncheck –pT stores_demo:monitoring

```
TBLspace Report for stores_demo:informix.monitoring

    Physical Address              1000f3
    Creation date                 09/12/2002 13:46:02
    TBLspace Flags                801         Page Locking
                                              TBLspace use 4 bit bit-maps

    Maximum row size              2008
    Number of special columns     0
    Number of keys                0
    Number of extents             2
    Current serial value          10
    First extent size             8
    Next extent size              8
    Number of pages allocated     16
    Number of pages used          10
    Number of data pages          9
    Number of rows                9
    Partition partnum             1048806
    Partition lockid              1048806

    Extents
        Logical Page  Physical Page        Size
                   0        1030c5           8
                   8        1030d5           8

    Type                 Pages      Empty  Semi-Full       Full  Very-Full
    ---------------- ---------- ---------- ---------- ---------- ----------
    Free                     6
    Bit-Map                  1
    Index                    0
    Data (Home)              9
                        ----------
    Total Pages             16

    Unused Space Summary

        Unused data slots                                      0

    Home Data Page Version Summary

                Version                                    Count

                      0 (oldest)                             8
                      1 (current)                            1
```

4(int) + 2000(char) + 4(int)

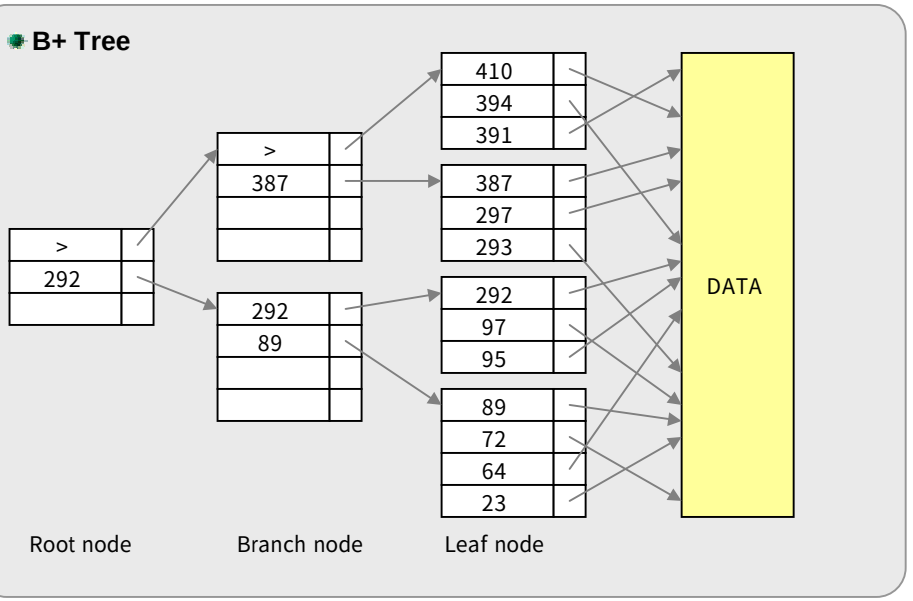
bitmap (1) + data (9)

B	D1	D8	D9
D2	D3	F	F
D4	D5	F	F
D6	D7	F	F

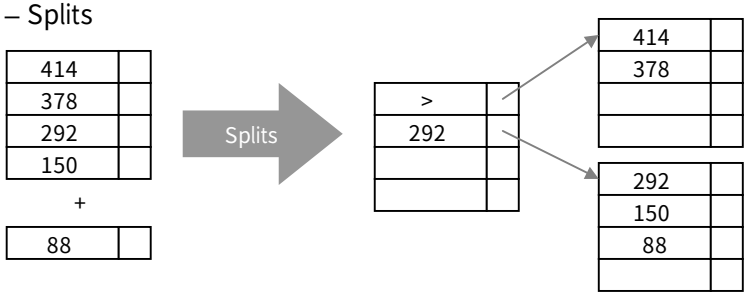
8건째 데이터 입력하면서 extent 추가 할당

테이블 alter 후 9번째 데이터 입력

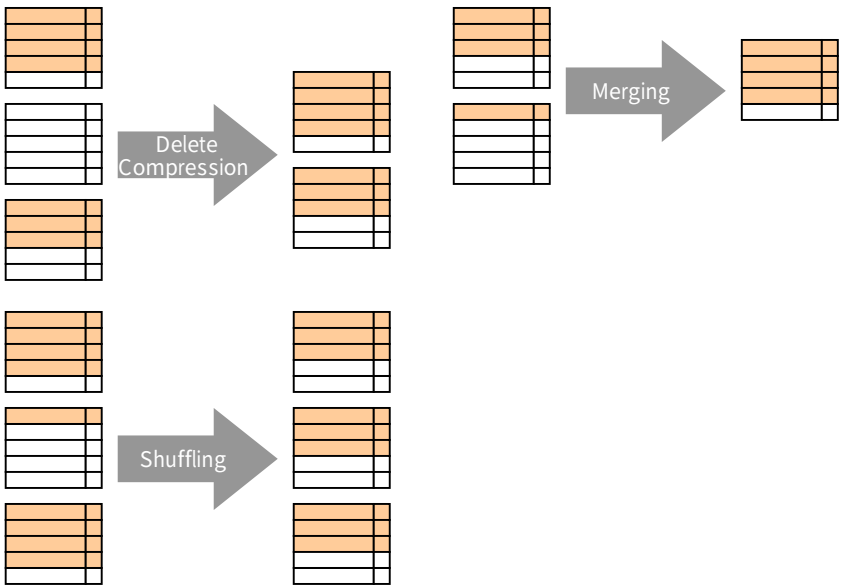
Index Structure



- 다음과 같은 이유로 인덱스 키는 작게 만드는 것이 바람직하다.
  - 하나의 페이지에 저장되는 인덱스 키가 많아진다
  - 인덱스 페이지의 노드 레벨이 감소한다
- 인덱스의 컬럼의 최대 크기는 390 바이트이다(9.3이후)
- 다차원(multi-dimension) 형태의 데이터에 대해서 R tree라는 인덱스 기법을 제공한다.
- 인덱스는 데이터의 변화에 따라 그 구조가 계속 변경된다.
  - Splits



  - 그밖에 Delete Compression, Merging, Shuffling 등이 있다



Creating / Altering / Dropping Indexes

- **인덱스 종류**
  - **Unique**
  - **Duplicate**
  - **Composite**
  - **Cluster**

- **인덱스 생성 기본 구문**

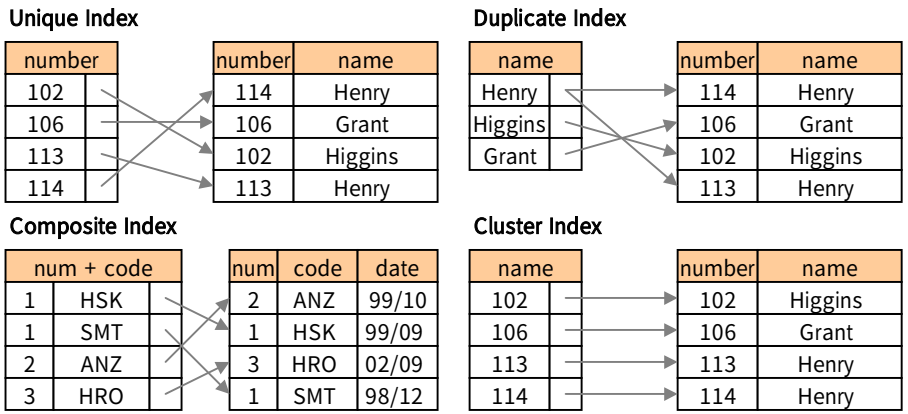
```
CREATE [ UNIQUE ] [ CLUSTER ] INDEX index_name
    ON table_name ( column_name_list )
    [ FILLFACTOR fillfactor_value ]
    [ IN dbspace ] ;
```

- **인덱스 변경**

```
ALTER INDEX index_name TO CLUSTER ;
RENAME INDEX old_index_name TO new_index_name ;
```

- **인덱스 삭제**

```
DROP INDEX index_name ;
```



- Composite 인덱스는 최대 16개 컬럼으로 구성될 수 있고 총 390바이트 크기까지 지정 가능 하다.
- Cluster 인덱스는 인덱스 순서에 맞게 데이터의 위치를 변경하는데, 한 테이블에는 하나의 cluster 인덱스만 의미를 가지게 된다. cluster 인덱스를 생성하는 시점에서만 데이터의 위치를 변경하므로 이후 insert나 update 가 있더라도 물리적인 데이터 위치를 인덱스 순으로 정렬하지는 않는다. 만약 다시 cluster 인덱스를 적용하려면 alter 구문을 사용해야 한다.

- 인덱스 생성 예제

```
CREATE UNIQUE INDEX ix_orders ON order (customer_num) ;  {Unique 인덱스 생성}
CREATE INDEX ix_items ON items (manu_code, stock_num) ;  {Composite 인덱스 생성}
CREATE INDEX CLUSTER is_manu ON manufact(manu_code);     {Cluster 인덱스 생성}
```

- 단일 컬럼에 대한 인덱스를 생성할 때는 오름차순(ASC,기본값), 내림차순(DESC)에 대한 구분을 할 필요가 없지만 composite 인덱스를 구성할 때 필요에 따라 DESC 키워드를 사용할 경우 효과적일 수 있다.

```
CREATE INDEX order_ix1 ON order (order_amt, order_date DESC) ;
```

- Detached 인덱스
  - 인덱스는 테이블의 데이터가 있는 dbspace에 생성되는데, 특정 dbspace에 생성되도록 지정할 수 있다

```
CREATE INDEX customer_ix ON customer (zipcode) in cust_ix_dbs ;
```

- Fillfactor는 인덱스를 생성할 때 인덱스 페이지를 기록하는 양을 정의한다.
  - Insert작업이 많은 경우 fillfactor를 낮추고, 데이터가 정적일 경우 fillfactor를 높인다. (기본값은 90%)

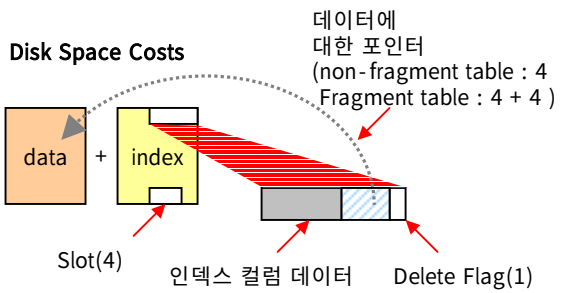
```
CREATE INDEX state_code_idx ON state(code) FILLFACTOR 80 ;
```

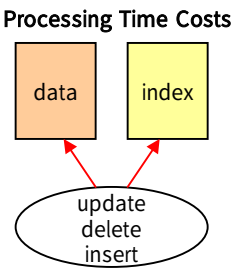
- 기타 구문

```
ALTER INDEX ix_man_cd TO CLUSTER ;
RENAME INDEX ix_cust TO new_ix_cust ;
DROP INDEX ix_stock ;
```

Indexing Guidelines

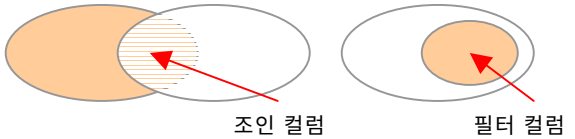
- 인덱스 사용의 장점
  - 데이터 페이지의 입출력 횟수가 감소한다
  - 인덱스 데이터에 대하여 정렬 작업이 생략된다
  - 데이터의 유일성을 보장할 수 있다 (Unique index)
  - Key-only 검색으로 데이터 페이지를 읽지 않을 수 있다
- 인덱스 소요 비용
  - 디스크 사용 공간
  - DML 문장 수행시간
- 인덱스 사용 가이드
  - 선택적인 필터 컬럼, 조인 컬럼에 사용한다
  - 정렬에 자주 사용되는 컬럼에 사용한다
  - 중복값이 많은 컬럼에는 사용하지 않는다
  - 주요 인덱스의 개수를 제한한다
  - 키값의 크기를 작게 한다
  - Composite 인덱스로 유일성을 증가시킨다
  - Cluster 인덱스로 데이터 조회 속도를 증가시킨다
  - 대량의 update, delete, insert 작업 전에 미리 인덱스를 disable한다



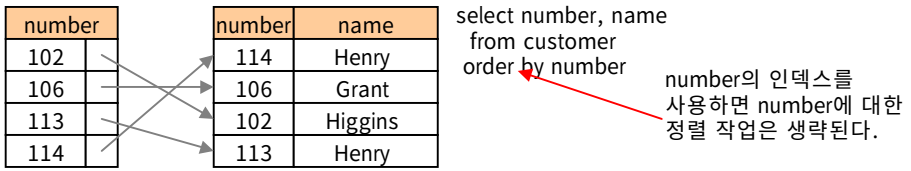


Cost of Indexing

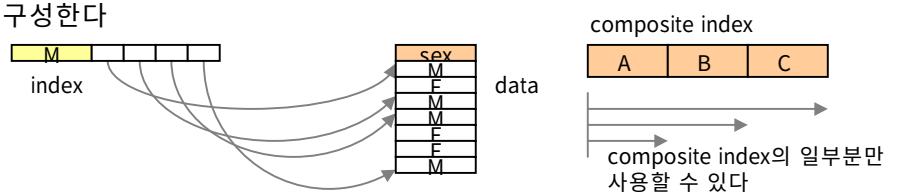
- 조인 컬럼에 인덱스가 없으면 자동적으로 temporary index를 생성하여 조인하거나 테이블을 sequential scan하여 hash 조인을 하게 된다.
- DSS환경에서는 인덱스를 사용한 조인보다 hash조인이 더 바람직 할 수 있다



- 인덱스 자체는 데이터가 정렬되어 있는 상태이다



- 중복값이 많은 데이터의 인덱스는 데이터 조회상에도 그다지 이점이 없고 DML 작업 중 부하가 크므로 가능하면 다른 컬럼과 함께 composite 인덱스를 구성한다

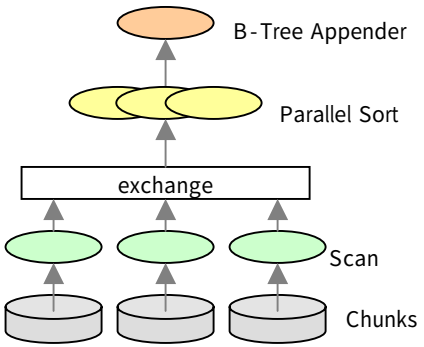


- 데이터가 빈번하게 변경되는 테이블은 과중한 인덱스가 오히려 성능을 저하시킬 수 있으므로 그 개수를 제한하는 것이 좋다. 또한 인덱스의 키가 작을 수록 인덱스 페이지를 조금만 사용하므로 성능이 향상된다.

- Cluster 인덱스를 생성하거나 변경하는 동안 데이터의 물리적인 순서가 해당 인덱스의 순서로 변경된다. cluster 인덱스를 통한 데이터 조회의 성능이 향상되지만 cluster 인덱스를 생성하는데 많은 시간이 소요된다.

Performance Tuning for Indexes

- **대량 데이터 로드 및 변경**
  - **Step 1 : Disable index**
  - **Step 2 : 로드 및 변경**
  - **Step 3 : Enable index**
- **병렬 인덱스 생성 (Parallel Index Build)**
  - **PDQPRIORITY > 0**
- **인덱스 메모리 상주 / 해제**
  SET INDEX index_name MEMORY_RESIDENT;
  SET INDEX index_name NON_RESIDENT;



Parallel Index Builds

- 인덱스 disable

```
SET INDEXES FOR table_name DIABLED ;
```

- 인덱스 enable

```
SET INDEXES FOR table_name ENABLED ;
```

- 병렬 인덱스 생성 단계
  - 1.scan 스레드가 디스크로부터 데이터 스캔
  - 2.데이터가 sort 스레드로 전달
  - 3.sort 스레드의 결과물이 b-tree appender로 전달되어 인덱스 생성
- PDQPRIORITY가 0일 경우 sort를 위한 메모리 할당 크기는 128K로 제한된다.
- 인덱스의 익스텐트 크기는 데이터 테이블의 크기에 비례하여 자동적으로 결정된다. unique 인덱스의 경우 대략 다음과 같이 익스텐트를 할당한다

$$\frac{\text{인덱스 row 크기}}{\text{테이블 row 크기}} \times \text{익스텐트 크기}$$

- 인덱스 메모리 상주 모니터링

```
Informix> dbaccess stores_demo << !
set index zip_ix memory_resident;
!
Informix> onstat –T
Tblspaces
 n address  flgs ucnt tblnum physaddr npages nused  npdata nrows  nextns resident
 1 b342018  0    1    100001 10000e   650    650    0      0      9      0
 2 b346d18  0    0    100002 10000f   4      4      2      12     1      0
                                      :
618 b4a9b90 2000 0    100272 10600c   4      2      0      0      1      2000
                                      :
Informix> dbaccess stores_demo << !
select zipcode from customer;
!
Informix> onstat –P
partnum  total    btree    data     other    resident dirty
0        4861     0        0        4861     0        0
1048577  2        0        0        2        0        0
1048578  5        1        2        2        0        0
                           :
1049202  3        1        0        2        1        0
```

```
dbaccess sysmaster << !
select hex(partnum),partnum
from systabnames
where dbsname = 'stores_demo'
and tabname = 'zip_ix'
!
(expression)    partnum
0x00100272      1049202
```

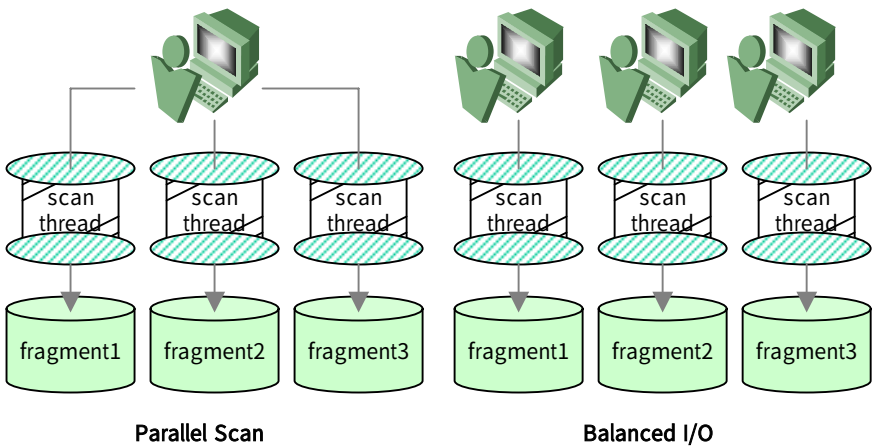
resident flag

resident된 페이지 수

Table Partitioning 0verview

- **Fragmentation (분할)**
  - **한 테이블의 데이터나 인덱스를 물리적으로 여러 dbspace에 저장하는 방식으로, 논리적인 테이블은 하나지만 그 테이블을 구성하는 tblspace는 여러 개가 된다.**

- **Fragment의 장점**
  - **병렬 스캔 등의 병렬 처리**
  - **Balanced I/O**
  - **백업 및 복구 단위 세분화 :**
    → 테이블의 데이터 중 영역별로 일부 데이터만 백업하거나 복구
  - **가용성 증가 (Higher Availability) : DATA SKIP**
    → 일부 영역이 접근 불가능한 상태라도 다른 영역에 대한 조회는 방해 받지 않으므로 DSS환경에서 대량의 데이터를 조회할 때 유용함
  - **영역별 보안 기능 추가**
    → Fragment단위로 사용 권한 부여 가능



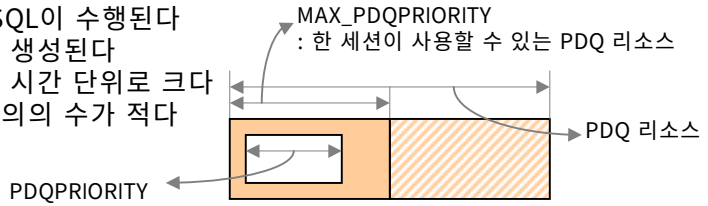
Parallel Scan Balanced I/O

- Fragment의 목표
  - 여러 디스크의 균형적인 I/O (balanced I/O) 구성
  - 불필요한 I/O를 제거하여 최대의 I/O 성능 (throughput) 보장

- 병렬 스캔 및 기타 병렬 처리 기능을 가능하게 하기 위해서 PDQ를 사용한다
  - 세션 환경 변수 : export PDQPRIORITY=40
  - 세션의 SQL문 : SET PDQPRIORITY 40;
- PDQ값은 전체 PDQ 리소스 중 사용할 양에 대한 비율을 나타낸다.
  - OFF 또는 0 : 병렬 기능 사용 안함
  - LOW 또는 1 : 병렬 스캔 기능만 동작
  - HIGH 또는 100 : PDQ 리소스 최대 사용
  - 2 ~ 99 : PDQ 리소스의 사용할 양 (%)
  - DEFAULT 또는 -1 : PDQPRIORITY 환경변수에 의해서만 결정

☞ PDQ에 관련한 자세한 내용은 IBM Informix Dynamic Server Self Study Guide for Server Administrator 를 참조한다.

- 병렬 스캔은 DSS 환경에서 필수적인데 DSS환경의 SQL은 다음과 같은 특징을 가지고 있다
  - 대량의 데이터를 읽고 트랜잭션은 거의 없거나 매주 적다
  - 데이터는 주로 sequential하게 읽는다
  - 복잡한(Complex) SQL이 수행된다
  - 대량의 임시 파일이 생성된다
  - 응답시간이 분 또는 시간 단위로 크다
  - 동시에 수행되는 질의의 수가 적다



- Balanced I/O는 OLTP환경에서 여러 사용자가 디스크에 대한 경합(contention)없이 동일한 테이블을 접근할 수 있게 하는 요소이다.
- OLTP환경에서의 SQL은 다음과 같은 특징을 가지고 있다
  - 상대적으로 적은 데이터를 조회한다.
  - 트랜잭션이 빈번하다
  - 주로 인덱스를 통한 스캔을 한다
  - 단순(Simple) SQL이 많다
  - 응답시간이 수초에서 밀리세컨드에 이르도록 한다
  - 동시에 수행되는 질의가 많다
- OLTP환경에서는 PDQPRIORITY를 반드시 0으로 설정해야 한다.

Fragmenting a Table / Index

- **Round Robin 분할**

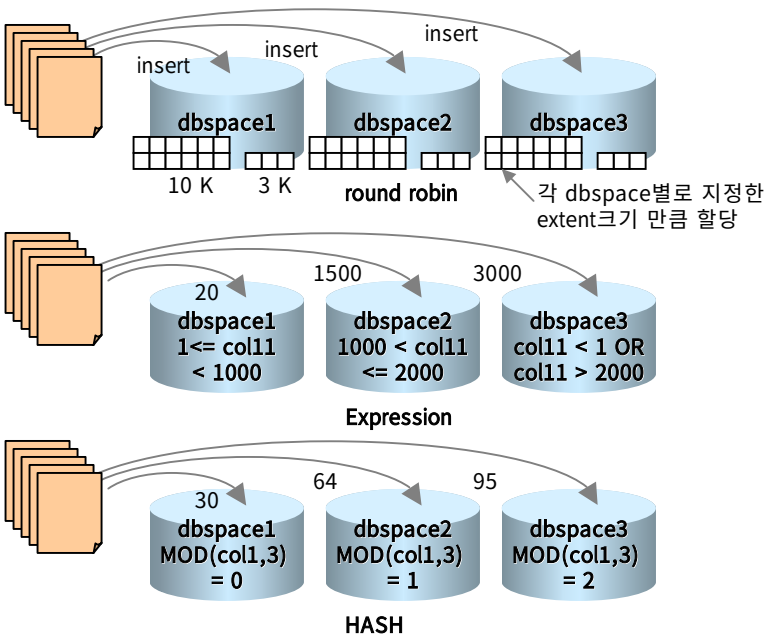
```
CREATE TABLE table1 ( … )
FRAGMENT BY ROUND ROBIN IN dbspace1, dbspace2,dbspace3
EXTENT SIZE 10000 NEXT SIZE 3000;
```

- **Expression 분할**

```
CREATE TABLE table2 ( col1 SERIAL, col2 CHAR(20), … )
FRAGMENT BY EXPRESSION
  col1 <= 1000 AND col11 >= 1 IN dbspace1,
  col1 <= 2000 AND col11 > 1000 IN dbspace2,
  REMAINDER IN dbspace3;
```

- **Hash 분할**

```
CREATE TABLE table3 (col1 SERIAL, col2 CHAR(20), … )
FRAGEMENT BY EXPRESSION
  MOD(col1, 3) = 0 IN dbspace1,
  MOD(col1, 3) = 1 IN dbspace2,
  MOD(col1, 3) = 2 IN dbspace3 ;
```



- **Round Robin 분할**
  - 데이터가 어느 dbspace에 저장되는지 알 수 없다.
  - 데이터에 의해 저장될 위치를 결정하지 않으므로 데이터가 수정될 때 다른 영역으로 이동되는 상황이 발생하지 않는다.
  - 빠른 데이터 로딩, 업데이트가 과중한 환경, sequential scan이 많은 환경에서 적합한 fragment 전략이다.

- **Expression 분할**
  - SQL expression에 의해 데이터가 저장될 dbspace가 결정된다
  - SQL expression에 해당 사항 없는 데이터는 REMAINDER IN 구문에서 지정한 dbspace로 저장된다
  - SQL expression에는 remote 테이블이나 subquery, stored procedure는 사용할 수 없다
  - 일부 dbspace만 I/O 할 수 있도록 데이터의 조회 범위가 작은 질의가 많을 경우 유용한 fragment 전략이다.
  - 데이터에 대한 expression을 검증해야 하므로 CPU 리소스가 round robin에 비해 더 필요하다.
  - 최대 2048개의 fragment 정의가 가능하고 SQL expression에는 다음과 같은 관계형, 또는 논리 연산자가 사용 가능하다
    - >, <, >=, <=, IN, BETWEEN
    - AND, OR

```
CREATE TABLE table4 (
    col1 SERIAL,
    col2 CHAR(20),
    … )
FRAGEMENT BY EXPRESSION
    col1 IN (101,203,420,532) IN dbspace1,
    col1 = 492 OR col2 = 'GA' IN dbspace2,
    col1 < 20000 IN dbspace3,
    col1 BETWEEN 20000 AND 40000 IN dbspace4,
    REMAINDER IN dbspace5 ;
```

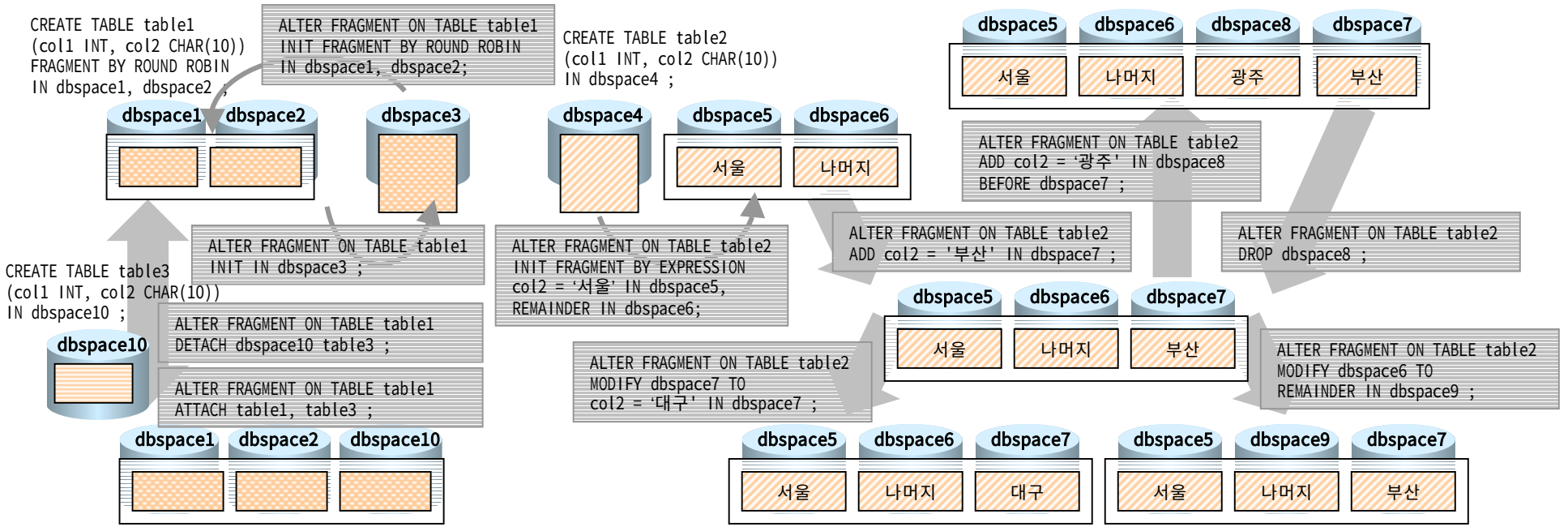
- **Hash 분할**
  - 데이터에 따라 고르게 fragment를 구성할 수 있다.
  - 전체 데이터에 대한 분포를 알 수 없는 경우 expression을 만들 수는 없지만 hash함수를 통해 데이터에 따라 저장될 dbspace를 알 수 있다

Fragmenting a Table / Index

- **ALTER FRAGMENT**
  - **Fragment 초기화**
    ALTER FRAGMENT ... INIT ...
  - **Fragment 추가**
    ALTER FRAGMENT ... ADD ...
  - **Fragment 삭제**
    ALTER FRAGMENT ... DROP ...
  - **Fragment expression 또는 dbspace 변경**
    ALTER FRAGMENT ... MODIFY ...
  - **Fragment 를 단일 테이블로 분리하거나 결합**
    ALTER FRAGMENT ... ATTACH ...
    ALTER FRAGMENT ... DETACH ...

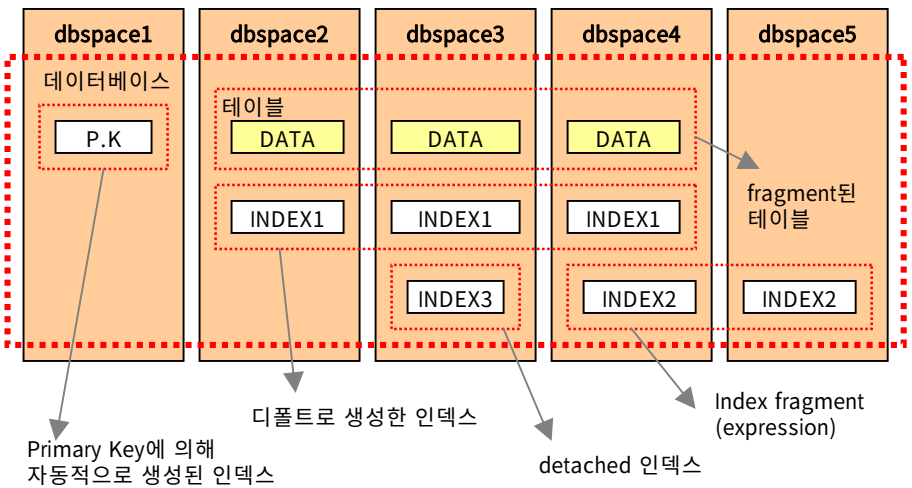
- Logging상태의 데이터베이스에서 Alter fragment를 사용하면 그 구문은 하나의 트랜잭션으로 처리된다.
  - 전체 테이블에 lock을 건다
  - 데이터 행의 fragment 위치가 변경되면 먼저 행이 삭제되고 새로운 위치로 추가된다.
  - 트랜잭션으로 처리되므로 대량의 테이블에 대해서 작업을 할 때는 no-logging상태로 변경하는 것이 좋다
- No-logging상태의 데이터베이스에서 alter fragment를 사용하면 테이블 전체에 lock을 걸고 새 fragment가 생성될 때까지 기존 fragment를 유지한다.
- Alter fragment를 수행하는 동안 기존 데이터와 alter후의 데이터를 동시에 가지고 있을 만큼 충분한 여유 공간이 있어야 한다.

- fragment를 drop하면 해당 데이터는 다른 fragment로 이동된다. expression의 경우 remainder fragment로 이동된다.
- fragment를 expression과 dbspace를 동시에 modify 할 수 없다.
- fragment된 테이블과 attach할 테이블의 스키마는 동일하여야 하고 저장된 dbspace는 서로 달라야 한다



Fragmenting a Table / Index

- **Index Fragmentation**
  CREATE INDEX idx1 ON table2 (col1)
  FRAGMENT BY EXPRESSION
  col1 < 1000 IN dbspace1,
  col1 >= 1000 IN dbspace2 ;
- **Index Non-fragment (detached Index)**
  CREATE INDEX idx1 ON table2 (col1) IN dbspace1;



- 인덱스의 저장위치는 테이블 데이터의 fragment 전략을 그대로 따르고 데이터가 저장되는 dbspace에 인덱스가 생성된다
  - Round Robin fragment 테이블의 인덱스 : round robin
  - Expression fragment 테이블의 인덱스 : expression
- 테이블의 fragment 전략과 다르게 인덱스 fragment를 결정하거나 인덱스는 fragment하지 않고 생성할 수 있다
- round robin 테이블에 인덱스를 생성하면 round robin으로 인덱스가 fragment 되지만 round robin으로 인덱스를 생성하는 구문을 사용할 수 없다.
  CREATE INDEX idx2 ON table2 (col1)
  FRAGMENT BY ROUND ROBIN IN dbspace1, dbspace2 ;
  -- syntax error
- Round Robin으로 fragment된 테이블에 unique 인덱스를 디폴트로 생성할 수 없고, 반드시 특정 dbspace에 생성하도록 detach 인덱스로 만들거나 expression으로 인덱스를 fragment 하여야 한다

	Round Robin 테이블	Expression 테이블
Dup.인덱스(or +detach)	○	○
Round Robin 인덱스	×	×
Expression 인덱스	○	○
Primary Key	○	○
Unique 인덱스	×	○ (테이블과 인덱스 expression에 사용된 컬럼이 같을 때)
Unique+detach	○	○
Unique+Expression	○	○

- Primary key나 foreign key를 만들 때 시스템에서 자동적으로 생성되는 시스템 인덱스의 경우 테이블의 fragment와 상관없이 데이터베이스가 생성된 dbspace에 인덱스가 생성된다.
- Fragment 테이블이 attach 될 때 다음과 같이 새 fragment가 기존 fragment에 대해 완전히 대칭적(symmetric)이라면 인덱스를 re-build 하지 않는다.
  - 데이터의 중복(data overlap)이 없다
  - 인덱스는 동일한 컬럼으로 구성된다.
  - 인덱스의 특성이 동일하다 (unique, duplicate)
  - 인덱스가 위치하는 dbspace가 중복되지 않는다.

Fragmenting Guideline

- **Fragment 전략 결정**
  - **데이터 access가 빈번한 테이블, 주요한 SELECT구문 선택**
  - **테이블 access 방법 분석**
  - **병렬적으로 사용할 수 있는 CPU와 디스크의 수 결정**
  - **업무상의 특징 적용**
  - **테이블, dbspace별로 I/O 검사**
    SET EXPLAIN, onstat –g ppf, onstat –g iof 등

- **Expression fragment guideline**
  - **REMAINDER IN 절을 피할 수 있도록 한다**
  - **데이터 분산이 아닌, I/O의 분산이 이뤄지도록 한다**
  - **Expression은 단순해야 한다**
  - **Expression에서 데이터 형 변환이 일어나지 않게 한다**
  - **보다 제한적인 표현식이 먼저 나올 수 있도록 한다**
  - **데이터 로딩 성능을 위하여 첫번째 expression 조건에 대부분 맞도록 한다**
  - **눈에 띄는 장점이나 효과가 없다면 fragment를 하지 않는다.**

• 테이블 access 방법 분석

Round Robin	Expression
Sequential 스캔	인덱스 스캔
	필터나 조인 컬럼으로 expression을 만들어 데이터를 고르게 분포 가능
DSS환경에서 적합하며, CPU와 fragment 개수가 증가하면 선형적으로 병렬 읽기 성능이 증가된다	OLTP환경에서 적합하다. Fragment의 목적이 조회할 데이터의 범위를 감소시키는데 있으므로 fragment 수의 증가와 성능 증가는 선형적이지 못하다

• 업무상의 주요 특징

Round Robin	Expression
데이터 로딩	fragment별로 사용 권한 부여

• REMAINDER fragment는 항상 조회하므로 REMAINDER 절을 사용하지 않도록 expression을 만든다.
• 복잡한 expression, 데이터 형 변환이 일어나는 경우 CPU 리소스를 더욱 사용하게 되므로 바람직 하지 않다.
• 다음과 같은 경우 25값을 입력하기 위해 6번 비교를 하게 된다.

```
x >= 1 AND x <= 10 IN dbspace1,
x >= 11 AND x <= 20 IN dbspace2,
x >= 21 AND x <= 30 IN dbspace3,
REMAINDER IN dbspace4
```

• 다음과 같이 expression을 고치면 4번 비교한다.

```
x <= 10 AND x >= 1 IN dbspace1,
x <= 20 AND x >= 11 IN dbspace2,
x <= 30 AND x >= 21 IN dbspace3,
REMAINDER IN dbspace4
```

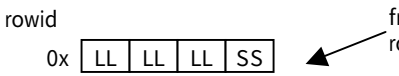
• 데이터 로딩은 round robin일 경우 성능이 우수하나 expression을 사용해야만 하는 경우 로드될 대부분의 데이터가 첫번째 expression에 만족하도록 만든다.

ROWIDS / Data Skip etc.

- **ROWIDS**
  CREATE TABLE table5 (...) WITH ROWIDS
  FRAGMENT BY ROUND ROBIN IN dbspace1, dbspace2 ;
  ALTER TABLE table4 ADD ROWIDS ;
  ALTER TABLE table4 DROP ROWIDS ;

- **Data Skip**
  - **SET DATASKIP ON ; -- dataskip 동작 시작**
  - **SET DATASKIP OFF ; -- dataskip 동작 종료**
  - **SET DATASKIP dbspace1 ; -- 특정 dbspace에 대한 dataskip**
  - **SET DATASKIP DEFAULT ; -- Config 파라매터에 의해 동작**

- **System catalog 정보**
  - **SELECT fragtype, a.tabid, tabname, strategy, dbspace**
    FROM sysfragments a, systables b
    WHERE a.tabid = b.tabid AND strategy in ('E','R') ;



fragment 된 테이블은 tblspace가 여러 개이므로 rowid가 중복된다

L : tblspace안의 논리적인 페이지 번호
S : slot 번호

※ Round Robin for Smart Blobs

```
Informix> dbaccess stores_demo << !
CREATE TABLE movie (
  movie_num INTEGER,
  movie_title CHAR(50),
  video BLOB,
  audio BLOB,
  description CLOB),
PUT video IN (sbsp3, sbsp6, sbsp7),
    audio IN (sbsp1, sbsp2, sbsp4),
    description IN (sbsp5);
!
```

- fragment되지 않은 테이블의 모든 데이터 행은 unique한 rowid를 가지고 있지만, fragment된 테이블의 rowid값은 unique하지 않다.
- 일부 클라이언트 프로그램에서는 데이터에 unique한 rowid를 필요로 하는 경우가 있는데, 이때 WITH ROWIDS 구문을 사용하면 4byte의 rowid컬럼이 각 데이터 행에 추가된다.

- 테이블의 데이터를 조회할 때 일부 fragment를 읽을 수 없는 상황에서도 전체 select구문이 에러처리 되지 않고 그 fragment만 읽지 않고 넘어갈 수 있도록 config 파라매터 중 DATASKIP인 SQL구문으로 DATASKIP을 trun on 시킨다.
- 일반적으로 DATASKIP은 DSS환경에서 유용하다.
- Fragment가 skip되더라도 SQLCA구조체에서 sqlca.sqlwarn.sqlwarn7 플래그는 'W'로 표시된다. (ESQL/C, ESQL/COBOL)
- 다음과 같은 상황에서는 DATASKIP이 동작할 수 없다
  - Referential Integrity : Parent-Child관계의 테이블에 데이터를 delete하거나 insert할 때 어느 한쪽의 fragment가 access불가능한 상태
  - Update : 수정된 행이 access불가능한 fragment로 이동되려 할 때
  - Insert/Delete : access불가능한 fragment로 insert나 delete할 때
  - Indexes : 인덱스 키값이 access불가능한 fragment에 저장되어야 할 때
  - Serial Key : 첫번째 fragment에 current serial 값을 저장하고 있는데, 첫번째 fragment가 access불가능하고 serial값을 insert 하려 할 때

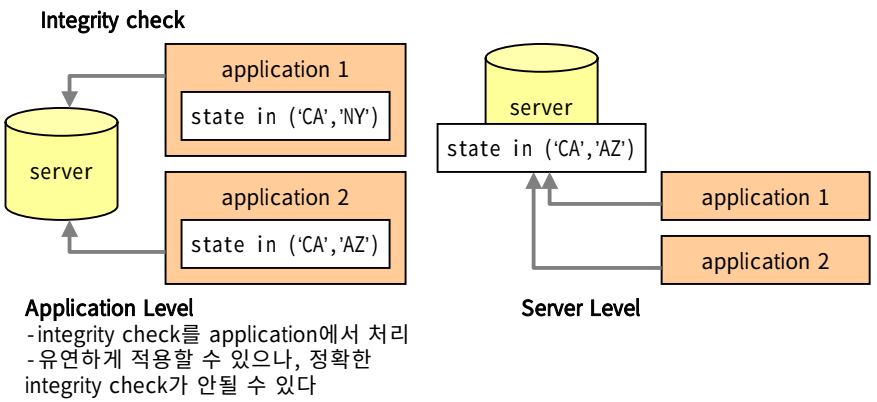
```
Informix> dbaccess stores_demo << !
SELECT fragtype, a.tabid, tabname, strategy, dbspace
  FROM sysfragments a, systables b
 WHERE a.tabid = b.tabid AND strategy in ('E','R') ;
!
   :
fragtype  T          <-- Table
tabid     238
tabname   frag1
strategy  R          <-- Round Robin
dbspace   ngn_data02

fragtype  I          <-- Index
tabid     238
tabname   frag1
strategy  E          <-- Expression
dbspace   ngn_data01
   :
```

3. Data Control

Integrity
Referential Constraints
Methods for Adding Constraints
Constraints Transaction-Modes
Drop Constraints, Etc..
Database Object Modes
Disabling / Enabling a Database Object
Filtering Mode & Recording Violations
Read Concurrency Control (Isolation)
Locks and Concurrency
Levels of Data Security
Granting / Revoking the Privileges
Using Roles
Views

Integrity

- **Integrity는 데이터베이스 안에서 데이터의 정확성을 의미한다**
  - **Referential Integrity : 테이블간의 관계(relationship)**
    - **Primary Key / Foreign Key**
  - **Entity Integrity : unique한 데이터 식별자 포함**
    - **Primary Key (unique identifier), UNIQUE**
  - **Semantic Integrity : 정확한 컬럼 정의**
    - **Data type, DEFAULT, CHECK, NOT NULL**



Application Level
- integrity check를 application에서 처리
- 유연하게 적용할 수 있으나, 정확한 integrity check가 안될 수 있다

Server Level

- Integrity를 정의하면서 데이터베이스에서 unique한 이름을 부여할 수 있는데, 이 이름을 constraint name이라 한다. Constraint name은 integrity 정의를 drop하거나 disable, enable할 때 사용된다.

```
CREATE TABLE orders (
    order_num INTEGER UNIQUE CONSTRAINT order_num_uq,
    order_date DATE NOT NULL CONSTRAINT order_date_nn DEFAULT TODAY) ;
ALTER TABLE orders
    MODIFY order_num INTEGER NOT NULL CONSTRAINT order_num_nn;
```

constraint name

- Constraint name을 지정하지 않으면 서버가 자동적으로 constraint ID 코드, table ID, constraint ID를 조합하여 생성한다. 하지만, 사용자가 의미를 직관적으로 알 수 있도록 부여하는 것이 바람직하다.

- Unique constraints는 내부적으로 unique 인덱스가 생성된다. 마찬가지로 Primary key와 Foreign key를 정의하면 자동적으로 각각 unique 인덱스와 duplicate 인덱스가 생성되는데, 만약 기존에 존재하는 인덱스가 있다면 그 인덱스를 사용하여 primary key또는 foreign key 속성을 가지게 만든다.

- Check constraints는 해당 컬럼에 insert되거나 update될 값의 범위 및 데이터를 제한한다

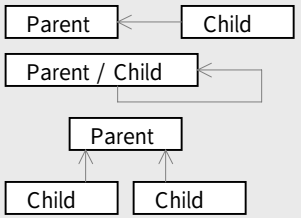
- Not null constraint는 지정한 컬럼에 null값을 허용하지 않으므로 insert나 update할 때 반드시 null값 외의 데이터를 저장해야 한다.

- 데이터를 입력할 때 컬럼 리스트에 지정되지 않은 컬럼에는 null값이 입력되는데, default constraint는 이러한 상황에서 자동적으로 입력될 값을 정의해 놓는다. 하지만 명시적으로 null을 입력할 경우에는 default 값이 입력되지 않는다.

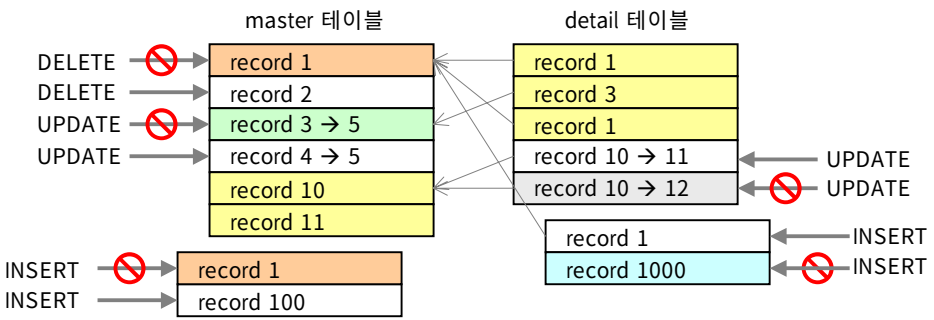
Referential Constraints

- **Referential Constraint는 primary key와 foreign key를 정의함으로써 테이블의 parent-child (또는 master-detail)관계를 만들어 준다.**
- **Referential Constraint를 정의하기 위해서는 테이블의 소유자이거나 REFERENCES 권한이 필요하다.**

- **Referential Constraint의 종류**
  - **Cyclic Referential Constraint**
  - **Self Referential Constraint**
  - **Multiple Path Constraint**



Referential Constraints의 성격



- **Cyclic Referential Constraints**는 두 개의 테이블간에 parent-child (혹은 master-detail)관계를 정의한다.

```
CREATE TABLE customer (
    customer_num SERIAL,
    fname CHAR(20),
    PRIMARY KEY (customer_num) CONSTRAINT pk_cnum );
CREATE TABLE orders (
    order_num SERIAL,
    customer_num INTEGER,
    FOREIGN KEY(customer_num) REFERENCES customer CONSTRAINT fk_cnum );

INSERT INTO customer VALUES (1,"Smith") ;
INSERT INTO orders VALUES (0, 1) ;
INSERT INTO orders VALUES (0, 2) ;
# 691: Missing key in referenced table for referential constraint (fk_cnum).
# 111: ISAM error: no record found.
DELETE FROM customer WHERE customer_num = 1 ;
# 692: Key values for constraint (pk_cnum) is still being referenced.
```

- **Self Referencing Referential Constraints**는 한 테이블 안에서 parent-child (혹은 master-detail)관계를 정의한다.

```
CREATE TABLE employee (
    enum SERIAL, mnum INTEGER,
    PRIMARY KEY (enum) CONSTRAINT pk_enum,
    FOREIGN KEY (enum) REFERENCES employee (enum) CONSTRAINT fk_enum ) ;

INSERT INTO employee VALUES (1,1);
INSERT INTO employee VALUES (2,1);
INSERT INTO employee VALUES (3,10);
# 691: Missing key in referenced table for referential constraint (fk_cnum).
# 111: ISAM error: no record found.
```

- **Multiple Path Referential Constraints**는 여러 테이블이 foreign key를 가지고 한 테이블의 primary key를 참조하는 형태 이다.

```
CREATE TABLE stock (
    stock_num SMALLINT,
    manu_code CHAR(3),
    PRIMARY KEY (stock_num,manu_code)
    CONSTRAINT pk_stock );
CREATE TABLE items (
    item_num SMALLINT,
    stock_num SMALLINT,
    manu_code CHAR(3),
    FOREIGN KEY (stock_num,manu_code)
    REFERENCES stock CONSTRAINT fk1 );

CREATE TABLE catalog (
    catalog_num SMALLINT,
    stock_num SMALLINT,
    manu_code CHAR(3),
    FOREIGN KEY (stock_num,manu_code)
    REFERENCES stock CONSTRAINT fk2 );
```

Referential Constraints

- **Cascading Deletes : Referential Constraint가 적용된 테이블에서 parent 테이블의 데이터 행을 삭제할 때 자동적으로 이 행을 참조하고 있는 child 테이블의 데이터 행을 삭제하는 기능으로 다음과 같은 조건을 만족해야 한다.**
  - **로깅 모드 데이터베이스에서 사용**
  - **Child 테이블이 DELETE의 correlated subquery에 포함 할 수 없다**
- **Primary key를 정의할 때 자동적으로 생성되는 unique 인덱스는 데이터베이스가 생성된 dbspace에 위치하게 된다.**

- Cascade delete는 한 트랜잭션으로 처리되므로 parent 테이블 또는 child 테이블에서 delete가 실패하면 전체 delete 구문이 rollback 된다.

```
CREATE TABLE customer (customer_num INT, PRIMARY KEY(customer_num));
CREATE TABLE orders (order_num INT, customer_num INT,
    FOREIGN KEY (customer_num) REFERENCES customer ON DELETE CASCADE);
```

- 데이터베이스가 로깅 모드가 아니면 cascade delete가 동작하지 않고 다음과 같은 correlated subquery가 포함되면 에러가 발생한다.

```
delete from customer  where customer_num >
(select min(customer_num) from orders where customer_num = customer.customer_num)
# 735: Cannot reference table that participates in cascaded delete.
```

- 이미 foreign key가 존재하는 테이블에 cascade delete 속성을 추가하려면 기존의 foreign key constraints를 삭제하고 새로 추가 해야 한다.

```
ALTER TABLE orders
DROP CONSTRAINT order_fk1,
ADD CONSTRAINT (FOREIGN KEY(customer_num)
REFERENCES customer ON DELETE CASCADE CONSTRAINT order_fk1) ;
```

- 만약 primary key에 의해 생성되는 unique 인덱스를 fragment하거나 다른 dbspace에 저장하고 싶다면 다음과 같은 순서로 수행 한다.
  - 테이블 생성
  - Unique 인덱스 생성
  - 테이블에 primary key 속성 부여

```
CREATE TABLE new_table ( col1 int, col2 char(20)) ;
CREATE UNIQUE INDEX new_index ON new_table(col1) IN dbspace10;
ALTER TABLE new_table ADD CONSTRAINT PRIMARY KEY (col1) CONSTRAINT new_pk;
```

Methods for Adding Constraints

- 테이블 생성시 추가하는 경우
  - 테이블 레벨 (Method 1)
  - 컬럼 레벨 (Method 2)
- 테이블 변경시 추가하는 경우
  - 테이블 레벨 (Method 3)
  - 컬럼 레벨 (Method 4)

• Constraint는 테이블 또는 컬럼 단위로 정의된다. 컬럼 단위의 constraint의 경우 다른 컬럼을 참조할 수 없다.

```
-- 테이블 단위의 constraints
ALTER TABLE items ADD CONSTRAINT CHECK (quantity >= 1 AND quantity <= 10 );
-- 컬럼 단위의 constraints
ALTER TABLE items MODIFY quantity SMALLINT CHECK (quantity < 10);
-- 컬럼 단위의 constraints에서는 다른 컬럼을 참조할 수 없다
ALTER TABLE orders MODIFY paid_date DATE CHECK (paid_date > ship_date);
#^
# 676: Invalid check constraint column.
-- 테이블 단위의 constraints에서만 컬럼간 참조가 가능하다
ALTER TABLE orders ADD CONSTRAINT CHECK(paid_date > ship_date);
```

• Method 1

```
CREATE TABLE customer (
    customer_num    SERIAL,
    fname           CHAR(20),
    PRIMARY KEY (customer_num) CONSTRAINT pk_cnum);
CREATE TABLE orders (
    order_num       SERIAL,
    customer_num    INTEGER ,
    FOREIGN KEY (customer_num) REFERENCES customer CONSTRAINT fk_cnum);
```

• Method 2

```
CREATE TABLE customer (
    customer_num    SERIAL   PRIMARY KEY CONSTRAINT pk_cnum,
    fname           CHAR(20) );
CREATE TABLE orders (
    order_num       SERIAL,
    customer_num    INTEGER REFERENCES customer CONSTRAINT fk_cnum );
```

• Method 3

```
ALTER TABLE customer ADD CONSTRAINT
    PRIMARY KEY (customer_num) CONSTRAINT pk_cnum ;
ALTER TABLE orders ADD CONSTRAINT
    FOREIGN KEY (customer_num) REFERENCES customer CONSTRAIN fk_cnum ;
```

• Method 4

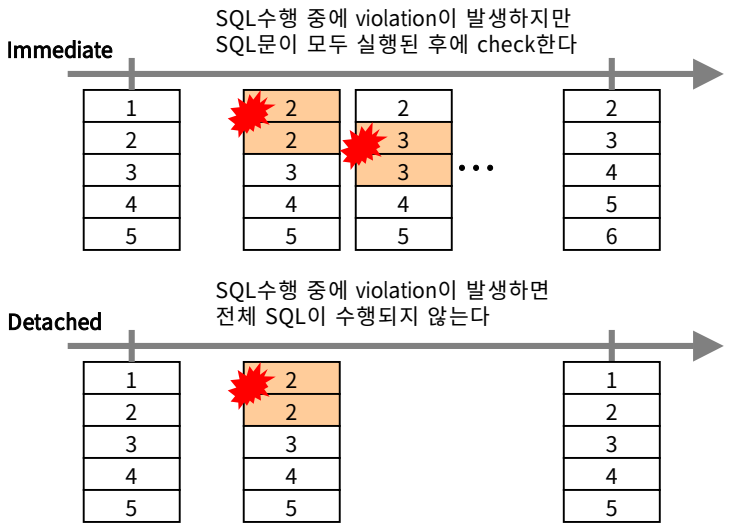
```
ALTER TABLE customer MODIFY customer_num SERIAL
    PRIMARY KEY CONSTRAINT pk_cnum ;
ALTER TABLE orders MODIFY customer_num INTEGER
    REFERENCES customer CONSTRAIN fk_cnum ;
```

Constraints Transaction-Modes

- **Constraints Transaction-Mode는 데이터에 해당하는 constraint를 언제 check할 것인지 결정한다**
  - **Immediate Constraint Checking : SQL문 끝에서 검사**
  - **Deferred Constraint Checking : Commit 시점에 검사**
  - **Detached Constraint Checking : SQL실행 중 각 행마다 검사**

- 로깅 모드의 데이터베이스를 사용할 때 기본 Constraints Transaction mode는 Immediate Constraint Checking이고 로깅 모드가 아닌 데이터베이스에서는 Detached Constraint checking이다.

```
CREATE TABLE immediate (current_no INTEGER UNIQUE );
INSERT INTO immediate VALUES (1) ; -- 1~5까지 입력
UPDATE immediate SET current_no = current_no + 1;
```



- 트랜잭션을 사용할 때 deferred constraint checking을 하게 되면 트랜잭션이 끝나는 시점에 모든 violation을 check한다.

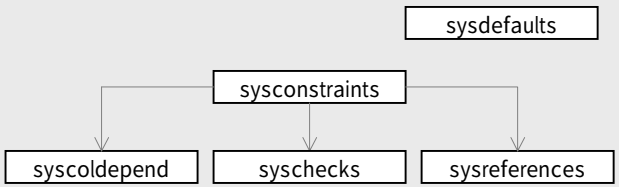
```
ALTER TABLE orders ADD CONSTRAINT PRIMARY KEY (order_num) ;
ATLER TABLE items ADD CONSTRAINT FOREIGN KEY (order_num) REFERENCES orders;

BEGIN WORK;
SET CONSTRAINTS ALL DEFERRED ;
UPDATE orders SET order_num = 1006 WHERE order_num = 1001;
UPDATE items SET order_num = 1006 WHERE order_num = 1001;
COMMIT WORK;
```

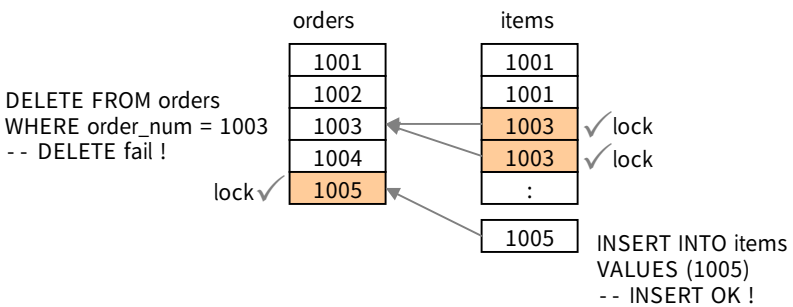
Drop Constraints, Etc..

- **Primary Key constraint를 삭제하면 이와 관련된 foreign key constraint도 함께 삭제된다**

- **Parent 테이블의 primary key를 삭제 또는 갱신하면 child 테이블의 foreign key 인덱스에 shared lock을 건다. 반대로 child 테이블에 데이터를 입력하거나 갱신하면 parent테이블의 primary key 인덱스에 lock을 거는데, 이러한 인덱스 lock은 데이터의 integrity를 check하는데 사용된다.**

- **System catalog table**





- constraint가 정의된 컬럼을 drop하면 관련된 constraint도 drop된다.

```
CREATE TABLE orders (
    order_num   SERIAL,
    order_date  DATE,
    PRIMARY KEY (order_num) CONSTRAINT pk_orders );
CREATE TABLE items (
    item_num    SERIAL,
    order_num   INTEGER,
    FOREIGN KEY (order_num) REFERENCES orders CONSTRAINT fk_orders );
ALTER TABLE orders DROP CONSTRAINT pk_orders;
-- 또는
ALTER TABLE orders DROP order_num;
```

constraint 동시 삭제

- system catalog 조회

```
CREATE TABLE test (
    col1 INT NOT NULL CONSTRAINT c_notnull,
    col2 INT DEFAULT 10,
    col3 INT UNIQUE CONSTRAINT c_unique,
    col4 INT CHECK (col4 > 100) CONSTRAINT c_check );
CREATE TABLE parent (
    col1 INT PRIMARY KEY CONSTRAINT parent_pk) ;
CREATE TABLE child (
    col1 INT,
    col2 INT REFERENCES parent(col1) CONSTRAINT child_fk);
```

```
SELECT constrid, constrname, tabname, constrtype
  FROM sysconstraints a, systables b
 WHERE a.tabid = b.tabid ;
```

constrid	constrname	tabname	constrtype
1	c_unique	test	U
2	c_notnull	test	N
3	c_check	test	C
4	parent_pk	parent	P
5	child_fk	child	R

```
SELECT constrid, tabname, colname
  FROM syscoldepend a, systables b, syscolumns c
 WHERE a.tabid = b.tabid AND b.tabid = c.tabid AND c.colno = a.colno ;
```

constrid	tabname	colname
2	test	col1
3	test	col4

```
SELECT * FROM syschecks WHERE type='T'
```

constrid	type	seqno	checktext
3	T	0	(col4 > 100)

```
SELECT constrid, primary, tabname
  FROM sysreferences a, systables b
 WHERE a.ptabid = b.tabid
```

constrid	primary	tabname
5	4	parent

Database Object Modes

- **Database Object Mode는 다음 중 한가지 상태를 가진다**
  - **Enabled : 일반 상태(default mode)**
  - **Disabled : Constraint가 적용되지 않고 trigger, index를 사용 안함**
  - **Filtering (trigger제외) : 에러가 발생해도 트랜잭션이 rollback되지 않는다**
- **Object Mode를 사용하는 이유**
  - **Constraint, trigger, index 가 없을 때 data loading이 더 빠르다**
  - **Database Object를 삭제하고 다시 생성하는 것보다 disabling하고 enabling하는 것이 더 편하고 정확하다**
  - **Filtering 모드에서 constraint를 위반하였을 때 rollback하는 대신 위반사항을 skip하고 계속 진행할 수 있다**

- Database Object Mode가 적용되는 대상은 다음과 같다
  - Constraint : Unique/ Referential Constraint / Check / NOT NULL
  - Index
  - Trigger

- Disable 모드에서는 constraint, index, trigger가 동작되지는 않지만 시스템 카타로그 테이블에서 해당 database object가 삭제된 것이 아니다.
- Filtering 모드는 data object가 enable되어 있는 상태와 같지만 error가 발생하더라도 트랜잭션이 rollback 되지 않는다. Trigger는 filtering 모드를 적용할 수 없다

- 데이터를 빠르게 로드하기 위해서는
  - Database object를 disable
  - 데이터 로드
  - Database object를 다시 enable

  순서로 수행한다. 만약 로딩할 데이터의 integrity가 보장되지 않는다면
  - Constraint를 Filtering하고 index를 disable한다.
- 보통 여러 건의 데이터를 update, delete, insert할 때 constraint를 위배하는 데이터 행을 찾기가 쉽지 않지만 violation 작업을 통해 에러를 로깅하면 이러한 데이터를 violation 테이블에 저장하고 있게 된다.

•시스템 카타로그 테이블

sysobjstate	
objtype	CHAR(1)
owner	CHAR(8)
name	CAHR(18)
tabid	INT
state	CHAR(1)

objtype
C : Constraints
I : Index
T : Trigger
state
D : Disabled
E : Enabled
F : Filtering with no error
G : Filtering with error

```
SELECT decode(objtype,"C","Constraints",
                      "I","Index",
                      "T","Trigger") object_type,
       tabname,
       decode(state,"D","Disabled",
                    "E","Enabled",
                    "F","Filtering",
                    "G","Filter with Error") state
  FROM sysobjstate a, systables b
 WHERE a.tabid = b.tabid
```

sysviolations	
targttid	INT
viotid	INT
diatid	INT
maxrows	INT

```
SELECT a.tabname target, b.tabname vio, c.tabname dia
  FROM sysviolations t, systables a, systables b, systables c
 WHERE a.tabid = t.targettid
   AND b.tabid = t.viotid
   AND c.tabid = t.diatid
```

Disabling / Enabling a Database Object

- **각각의 database object 를 disable**
  - **SET CONSTRAINTS constraint_name DISABLED ;**
  - **SET INDEXES index_name DISABLED ;**
  - **SET TRIGGERS trigger_name DISABLED ;**
- **특정 테이블의 모든 database object 를 disable**
  - **SET CONSTRAINTS, INDEXES, TRIGGERS FOR table_name DISABLED ;**
- **각각의 database object 를 enable**
  - **SET CONSTRAINTS constraint_name ENABLED ;**
  - **SET INDEXES index_name ENABLED ;**
  - **SET TRIGGERS trigger_name ENABLED ;**
- **특정 테이블의 모든 database object 를 enable**
  - **SET CONSTRAINTS, INDEXES, TRIGGERS FOR table_name ENABLED ;**

- 기존에 생성되어 있는 database object뿐 아니라, 새로 생성하는 database object에 대해서도 disable 시킬 수 있다.

```
CREATE UNIQUE INDEX idx1 ON employee(emp_no) DISABLED ;
CREATE TABLE customer (
    customer_num SERIAL,
    state CHAR(2) CHECK(state IN ("CA","AZ")) DISABLED ) ;
CREATE TRIGGER t1 UPDATE ON orders
    BEFORE (EXECUTE PROCEDURE x1()) DISABLED ;
```

- 또한 ALTER TABLE 구문을 사용할 때도 filtering이나 disable모드로 constraint를 생성할 수 있다.

```
ALTER TABLE employee ADD CONSTRAINT
    CHECK (age < 100) CONSTRAINT age_limit FILTERING ;
```

- Database Object Mode를 disable 하거나 enable하는 단위는 다음과 같다
  - Object단위 : Constraint 이름,index 이름, trigger 이름을 명시해야 한다
    - Object의 이름은 dbschema를 이용하거나 시스템 카타로그 테이블의 sysconstraints를 조회하여 알 수 있다
  - 테이블단위 : 테이블에 정의된 모든 constraint, index, trigger를 한번에 disable 또는 enable할 수 있다
- Disable된 database object의 constraint는 체크되지 않고, trigger는 동작되지 않으며 index는 사용되지 않는다. 또한 optimizer가 해당 index를 query path에 참조하지도 않는다.
- Referential 또는 unique constraint를 정의하면서 자동으로 생성된 index는 constraint 의 mode와 동일하게 enable, disable된다.
- SET CONSTRAINT문장이 실행될 때 해당 테이블에 exclusive lock이 걸린다.

```
-- object 단위
SET CONSTRAINTS c117_11, c117_12 DISABLED ;
SET INDEXES idx_x1 DISABLED ;
SET TRIGGERS upd_cust DISABLED ;
-- 테이블 단위
SET CONSTRAINTS, INDEXES, TRIGGERS FOR customer DISABLED ;
```

- Constraint가 disable상태에서 enable상태가 될 때 모든 행에 대해 constraint 검사를 하고 constraint 위배사항이 발견되면 error를 리턴하고 constraint를 여전히 disable 상태로 남는다.
- Constraint가 filtering상태에서 enable될 때 모든 행은 이미 constraint 를 만족하는 상태이므로 constraint검사를 하지 않는다.
- Index를 disable상태에서 enable시킬 때 전체 index를 재생성하게 된다.

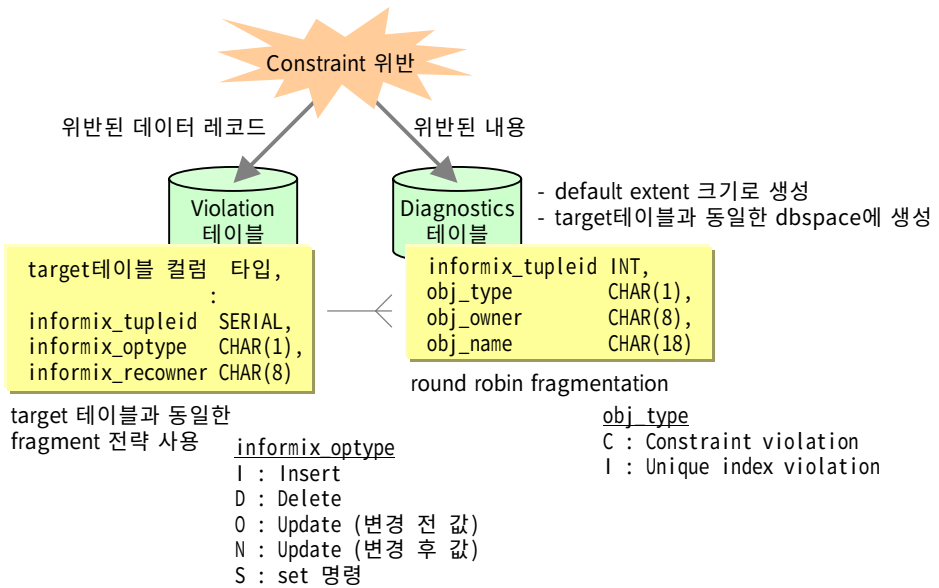
```
-- object 단위
SET CONSTRAINTS c117_11, c117_12 ENSABLED ;
SET INDEXES idx_x1 ENABLED ;
SET TRIGGERS upd_cust ENABLED ;
-- 테이블 단위
SET CONSTRAINTS, INDEXES, TRIGGERS FOR customer ENABLED ;
```

```
dbaccess stores_demo << !
SELECT constrname
  FROM sysconstraints a, systables b
 WHERE a.tabid = b.tabid
   AMD tabname = 'employee'
   AND constrtype = 'P'
!
```

```
dbaccess stores_demo << !
SELECT idxname
  FROM sysindexes a, systables b
 WHERE a.tabid = b.tabid
   AMD tabname = 'employee'
!
```

Filtering Mode & Recording Violations

- **Violation 테이블 구성 및 violation로깅 시작**
  - **START VIOLATIONS TABLE FOR target_table**
    [USING violation_table, diagnostics_table]
    [MAX ROWS number_of_diagnostics_per_violation] ;
- **Violation Logging 중단**
  - **STOP VIOLATIONS TABLE FOR target_table ;**
- **Filtering 모드**
  - **SET CONSTRAINTS constraint_name FILTERING**
    [WITH ERROR];
  - **SET INDEXES index_name FILTERING [WITH ERROR] ;**
  - **SET CONSTRAINTS, INDEXES**
    FOR table_name FILTERING [WITH ERROR] ;



- Database object가 enabled 모드로 전환되거나 filtering 모드에서 constraint를 위반하는 데이터를 violation테이블에, 그 위반 내역을 diagnostics 테이블에 기록한다. 이 두 테이블은 항상 쌍으로 구성된다.

- SART VIOLATIONS TABLE 구문을 실행시키면
  - violation테이블과 diagnostics테이블이 생성
  - Violation이 감지되면 테이블에 해당 데이터 기록
- MAX ROWS 옵션은 하나의 violation에 대하여 최대로 저장 가능한 diagnostics 데이터의 수를 나타낸다. (diagnostics 테이블의 총 데이터 수를 나타내는 것이 아님)
- START VIOLATIONS TABLE구문은 테이블의 소유자나 DBA권한을 가진 사용자가 실행할 수 있다
- USING절을 이용하여 테이블 이름을 직접 지정하지 않으면 자동적으로
  - target테이블이름_vio, target테이블이름_dia
  로 생성된다.
- 7.x의 경우 테이블이름의 최대 길이는 18자 이므로 target테이블 이름이 14자가 넘으면 USING절을 사용하여 violation 테이블 이름을 명시하는 것이 좋다 (9.x의 경우 테이블 이름은 최대 128자)

- Violation 로깅을 중단하더라도 생성된 violation 테이블이 자동으로 drop되지 않는다. violation 테이블이 존재하면 다시 START VIOLATIONS TABLE 구문을 실행할 수 없으므로 STOP VIOLATIONS TABLE을 한 후 DROP TABLE문으로 직접 삭제해야 한다.

- Database object를 Filtering 모드로 변경하더라도 START VIOLATIONS TABLE 구문으로 violation logging을 하지 않으면 filtering되지 않는다.
  896: Violations table is not started for the target table.
- Filtering 모드에서는 에러가 리턴되지 않는데, WITH ERROR구문을 사용하여 filtering 모드로 전환하면 violation을 로깅하면서 에러를 리턴한다.
  971: Integrity violations detected.
- 트랜잭션 중에 target테이블에 대한 rollback을 하게 되면 violation테이블과 diagnostics테이블도 함께 rollback된다.

Exercise 1

- 테이블 생성
- 테이블을 disabled상태로 변경
- Violation 로깅을 시작
- 데이터 입력
  INSERT INTO parent VALUES (0,'Donald');
  INSERT INTO child VALUES (0,100,'Frank');
- 테이블을 enabled 상태로 변경
- Violation 확인
- 위반 데이터 수정
- 테이블 enable

테이블 스키마

```
CREATE TABLE parent (
    p_no SERIAL NOT NULL CONSTRAINT n_1,
    name CHAR(15),
    PRIMARY KEY (p_no) CONSTRAINT p_1 ) ;
CREATE TABLE child (
    c_no SERIAL NOT NULL CONSTRAINT n_2,
    p_no INTEGER NOT NULL CONSTRAINT n_3,
    name CHAR(15)) ;
ALTER TABLE child ADD CONSTRAINT
    FOREIGN KEY (p_no) REFERENCES parent CONSTRAINT f_1 ;
```

※ Database Object를 disable할 때 순서도 중요하다

```
SET CONSTRAINTS, TRIGGERS, INDEXES FOR parent DISABLED;
# 892: Cannot disable object (informix.p_1) due to other active objects using it.
```

※ 마찬가지로 enable할 때도 순서에 주의해야 한다

```
SET CONSTRAINTS, TRIGGERS, INDEXES FOR child ENABLED;
#  893: Cannot activate/create object (informix.f_1) because of its dependencies.
```

• child테이블부터 disable한다

```
SET CONSTRAINTS, TRIGGERS, INDEXES FOR child  DISABLED;
SET CONSTRAINTS, TRIGGERS, INDEXES FOR parent DISABLED;
```

• Violation 로깅을 시작한다

```
START VIOLATIONS TABLE FOR parent;
START VIOLATIONS TABLE FOR child;
```

• Parent 테이블부터 enable한다.

```
SET CONSTRAINTS, TRIGGERS, INDEXES FOR parent ENABLED;
SET CONSTRAINTS, TRIGGERS, INDEXES FOR child  ENABLED;
#  971: Integrity violations detected.
```

• Database Object의 상태를 조회한다.

```
SELECT distinct tabname FROM sysobjstate a, systables b
 WHERE a.tabid = b.tabid
   AND state != 'E';
```

```
tabname  child
```

• Violation테이블을 조회한다.

```
SELECT * FROM child_vio a, child_dia b
 WHERE a.informix_tupleid = b.informix_tupleid;
```

```
c_no                1
p_no                100
name                Frank
informix_tupleid    1
informix_optype     S
informix_recowner   informix
informix_tupleid    1
objtype             C
objowner            informix
objname             f_1
```

• Child테이블을 수정하고 다시 enable 시킨다.

```
UPDATE child SET p_no = 1 WHERE c_no = 1;
SET CONSTRAINTS, TRIGGERS, INDEXES FOR child ENABLED;
```

• Child 테이블의 상태를 조회한다

```
SELECT name,
       decode(state,"D","Disabled",
                    "E","Enabled",
                    "F","Filtering",
                    "G","Filter with Error") state
  FROM sysobjstate a, systables b
 WHERE a.tabid = b.tabid
   AND tabname = 'child'
```

```
name      state

n_2       Enabled
n_3       Enabled
 146_55   Enabled
f_1       Enabled
```

Exercise 2

- **Parent, Child 테이블을 filtering 모드로 전환한다**
- **Child 테이블에 데이터 입력**
  INSERT INTO child VALUES (0,11,'Dick');
- **Parent, Child 테이블을 enabled 모드로 변경**
- **Violation 로깅을 중단**
- **위반 데이터 처리**
- **Violation 테이블 drop**

• Filtering 모드로 전환 (trigger는 제외)

```
SET CONSTRAINTS, INDEXES FOR parent FILTERING ;
SET CONSTRAINTS, INDEXES FOR child  FILTERING ;
```

• 데이터 입력 후 enable 모드로 전환

```
INSERT INTO child VALUES (0,11,'Dick');
-- 0 row(s) inserted.
SET CONSTRAINTS, INDEXES FOR child  ENABLED ;
SET CONSTRAINTS, INDEXES FOR parent ENABLED ;
```

• Violation 로깅을 중단하고 violation 테이블을 검색

```
STOP VIOLATIONS TABLE FOR parent;
STOP VIOLATIONS TABLE FOR child;
SELECT * FROM child_vio a, child_dia b
 WHERE a.informix_tupleid = b.informix_tupleid
   AND a.informix_tupleid > 1 ;
```

```
c_no                2
p_no                11
name                Dick
informix_tupleid    2
informix_optype     I
informix_recowner   informix
informix_tupleid    2
objtype             C
objowner            informix
objname             f_1
```

• Parent 테이블에 데이터를 입력하고 violation 테이블의 내용을 child 테이블로 입력한다.

```
INSERT INTO parent VALUES (11,"Marvin") ;
INSERT INTO child
SELECT c_no, p_no, name FROM child_vio WHERE informix_tupleid = 2 ;
```

• 더 이상 필요 없으면 violation 테이블을 삭제한다

```
DROP TABLE parent_vio;
DROP TABLE parent_dia;
DROP TABLE child_vio;
DROP TABLE child_dia;
```

Read Concurrency Control (Isolation)

- **ANSI SQL-92 Transaction Isolation (SET TRANSACTION)**
  - **Read Uncommitted**
  - **Read Committed**
  - **Repeatable Read**
  - **Serializable Read**
- **Informix Isolation (SET ISOLATION)**
  - **Dirty Read**
  - **Committed Read**
  - **Cursor Stability**
  - **Repeatable Read**
- **Access Mode**
  - **SET TRANSACTION READ WRITE ;**
  - **SET TRANSACTION READ ONLY ;**

ANSI Levels	SET TRANSACTION	SET ISOLATION
Read Uncommitted	READ UNCOMMITTED	DIRTY READ
Read Committed	READ COMMITTED	COMMITTED READ
N/A	N/A	CURSOR STABILITY
Repeatable Read	REPEATABLE READ	REPEATABLE READ
Serializable	SERIALIZABLE	REPEATABLE READ
비교	- ANSI표준 - access mode지원 - 트랜잭션당 한번만 - 트랜잭션이 끝날 때까지 유효 - 트랜잭션에서만 사용	- non-ANSI - 트랜잭션에서 전환가능 - 트랜잭션이 끝나거나 다른 SET ISOLATION구문을 실행할 때까지 유효 - 로깅모드 데이터베이스에서 언제든 사용 가능

- ANSI SQL-92표준의 SET TRANSACTION과 non-ANSI SQL인 SET ISOLTAION 구문 모두 사용 가능하다.
- SET TRANSACTION은 트랜잭션이 끝날 때, SET ISOLATION은 트랜잭션이 끝나거나 다른 SET ISOLATION 구문을 실행할 때까지 유효하게 적용된다.

```
BEGIN WORK;
SET TRANSACTION ISOLATION LEVEL
    READ UNCOMMITTED;
SELECT fname,lname FROM customer
 WHERE customer_num = 101;
SET TRANSACTION ISOLATION LEVEL
    READ COMMITTED;
COMMIT WORK;
SET TRANSACTION ISOLATION LEVEL
    READ UNCOMMITTED;
```

```
BEGIN WORK;
SET ISOLATION TO DIRTY READ ;

SELECT fname,lname FROM customer
 WHERE customer_num = 101;
SET ISOLATION TO COMMITTED READ ;

COMMIT WORK;
SET ISOLATION TO DIRTY READ ;
```

```
# 876: Cannot issue "Set Transaction" in an active transaction.
```

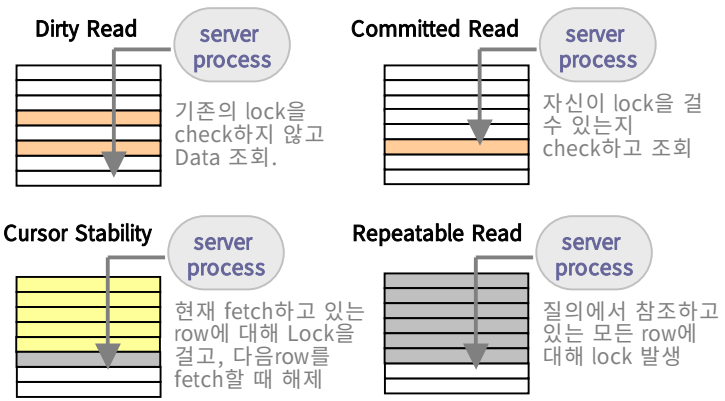
```
# 255: Not in transaction.
```

- Read-only 트랜잭션 모드 상태에서는 조회만 가능하고 다음과 같은 작업은 불가능하다
  - Update, insert, delete
  - Add, drop, alter, rename
  - Update statistics
  - Grant, revoke 등

```
BEGIN WORK;
SET TRANSACTION READ ONLY ;
UPDATE customer SET fname = 'new_name' WHERE customer_num = 101;
# 878: Invalid operation for a READ-ONLY transaction.
```

Read Concurrency Control (Isolation)

- **Read Uncommitted / Dirty Read**
  - **SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED ;**
  - **SET ISOLATION TO DIRTY READ ;**
- **Read Committed / Committed Read**
  - **SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;**
  - **SET ISOLATION TO COMMITTED READ ;**
- **Cursor Stability**
  - **SET ISOLATION TO CURSOR STABILITY ;**
- **Serializable / Repeatable**
  - **SET TRANSACTION ISOLATION LEVEL SERIALIZABLE ;**
  - **SET TRANSACTION ISOLATION LEVEL REPEATABLE READ ;**
  - **SET ISOLATION TO REPEATABLE READ ;**



- **Read Uncommitted / Dirty Read**
  - 데이터를 읽는 동안 lock을 걸지도 않고 lock을 체크하지도 않는다
  - Phantom row를 읽을 가능성이 있다
  - Phantom row는 트랜잭션에서 데이터가 변경되고서 나중에 rollback되어 데이터베이스 안에서는 존재하지 않는 값을 가진 경우의 데이터이다.
  - 다음과 같은 경우 유용하다
    - 데이터 변경이 없는 정적인 테이블
    - 100% 정확도 보다 빠른 응답이 중요한 경우
    - lock을 무시해야 하는 경우
  - 데이터베이스가 로깅 모드가 아닌 경우 Dirty read isolation만 적용된다.
- **Read Committed / Committed Read**
  - 로깅 모드의 데이터베이스의 기본 isolation 레벨.
  - 조회할 데이터에 대해 shared lock을 걸 수 있는지 검사를 먼저 하지만 실제로 lock을 걸지는 않는다. 즉 update, exclusive lock이 있는 데이터에 대해서는 읽을 수 없다. 그렇지만 lock을 걸지 않기 때문에 읽고 있는 데이터를 다른 사용자가 변경할 수 있다.
  - Commit된 데이터 만을 읽을 수 있기 때문에 Phantom row (또는 dirty data)를 읽지 않는 것을 보장한다.
- **Cursor Stability**
  - Cursor를 사용하여 데이터를 조회할 때만 설정할 수 있다
  - Cursor로 조회하고 있는 데이터행에 대하여 shared lock을 걸기 때문에 데이터를 조회하는 동안 다른 세션이 update, delete를 할 수 없다
  - Shared lock은 다음 행을 읽을 때까지만 유지되므로 현재 읽고 있지 않은 result set의 다른 행은 변경이 가능하다 .
- **Serializable / Repeatable Read**
  - ANSI 로깅 모드의 데이터베이스에서 기본 isolation 레벨.
  - 조회하는 모든 데이터에 대하여 shared lock을 건다. 그러므로 result set에 해당하는 데이터만 lock을 걸게 하려면 반드시 index scan을 하도록 해야 한다
  - Index스캔을 하지 않고 sequential 스캔을 하게 되는 경우 테이블 전체에 하나의 shared lock을 걸게 된다.
  - 데이터를 조회하는 동안 변경되지 않는 것이 보장되지만 동시성(concurrency)이 떨어진다.

```
Informix> onstat -g sql
Sess  SQL            Current            Iso Lock
Id    Stmt type      Database           Lvl Mode
215   SELECT         stores7            NL  Not Wait
209   SELECT         stores_demo        DR  Not Wait
207   SELECT         stores_demo        CR  Not Wait    0    0    9.03
```

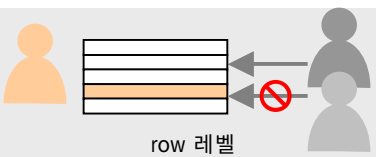
```
NL : No logging DB
DR : Dirty Read
CR : Committed Read
CS : Cursor Stability
RR : Repeatable Read
```

Locks and Concurrency

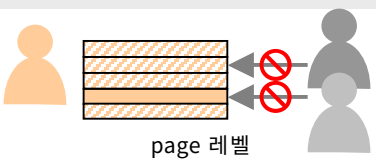
- **Lock의 적용 단위**
  - **Database 레벨**
  - **Table 레벨**
  - **Page 레벨**
  - **Row 레벨**
  - **Key 레벨**

- **Page 및 row 레벨 lock**
  - 테이블의 스키마 정의에 따라 결정된다.
  - Page 레벨 lock은 데이터 행이 포함된 page전체에 lock 걸리는 방식인데, 데이터가 sequential하게 저장되어 있다면 하나의 page lock으로 여러 개의 데이터 행에 대해 lock을 걸 수 있으므로 resource를 절약할 수 있 수 있으나, 각각의 데이터 행에 lock을 거는 row 레벨 lock에 비해 동시성이 떨어진다.

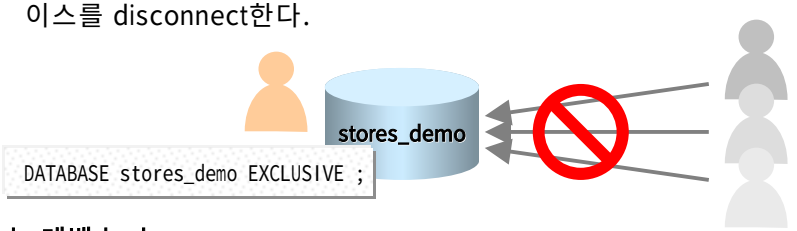
```
CREATE TABLE orders_new (
    order_num     SERIAL NOT NULL,
    customer_num INTEGER,
    order_date    DATE) LOCK MODE ROW ;
ALTER TABLE orders_new LOCK MODE (PAGE) ;
```



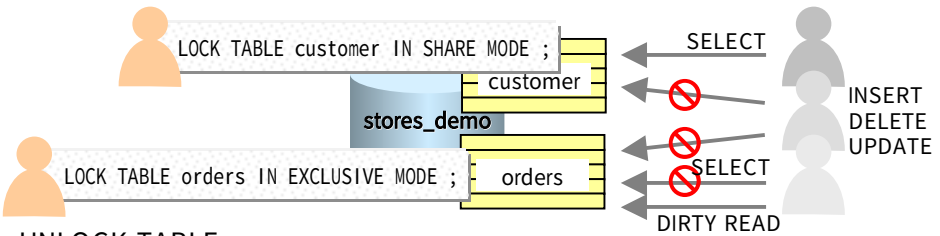
※ default lock 레벨은 page인데, 환경변수나 config 변수로 default lock 레벨을 row로 바꿀 수 있다. (9.30이상)
export IFX_DEF_TABLE_LOCKMODE=row



- **Database 레벨 lock**
  - 데이터베이스를 접속하면 자동적으로 shared lock이 걸린다.
  - 다른 사용자의 접근을 막고 데이터베이스에 대한 관리 작업을 할 때 exclusive lock을 걸 수 있다.
    - 대량의 테이블에 대한 변경 작업
    - 데이터베이스 백업을 위한 테이블 조회
    - 데이터베이스 구조 변경
  - exclusive lock을 해제하고 다른 사용자의 접근을 허용하려면 데이터베이스를 disconnect한다.



- **Table 레벨 lock**
  - SHARE MODE
    - 다른 세션은 테이블을 조회할 수 있지만 update, insert, delete는 불가능 하다.
    - share mode에서 데이터를 수정하면 그 데이터 행에 대해서는 exclusive lock이 걸린다.
  - EXCLUSIVE MODE
    - 다른 세션은 테이블을 조회 및 수정이 불가능하지만, dirty read isolation으로는 조회가 가능하다.



- UNLOCK TABLE
  - 로깅모드의 데이터베이스에서는 반드시 트랜잭션 안에서 사용해야 하고 트랜잭션이 종료되면 lock은 해제된다
  - 로깅 모드가 아닌 데이터베이스에서는 UNLOCK TABLE 구문을 사용하여 테이블의 lock을 해제한다.
    - UNLOCK TABLE table_name ;

Locks and Concurrency

- **Lock의 종류**
  - **Shared Lock**
  - **Exclusive Lock**
  - **Update Lock**
- **Lock 모드 설정**
  - **SET LOCK MODE TO NOT WAIT;**
  - **SET LOCK MODE TO WAIT 20;**
  - **SET LOCK MODE TO WAIT;**

• Lock의 종류
  – **Shared Lock**
    • 다른 세션에게 조회는 허용하고 데이터 변경은 허용하지 않는다.
    • Shared lock이 걸린 오브젝트에 다른 shared lock을 걸 수 있다
  – **Exclusive Lock**
    • 다른 세션은 데이터 변경 및 조회가 불가능 하다. (단, dirty read isolation 으로 조회하는 세션은 허용한다)
    • 관리 작업을 위하여 데이터베이스나 테이블에 의도적으로 lock을 걸 수 있다
  – **Update Lock (promotable lock)**
    • For update 커서를 사용할 때 발생한다
    • 데이터를 수정하기 위하여 조회하는 동안은 shared lock과 같이 다른 세션이 같은 데이터를 조회할 수 있지만, 데이터를 변경하는 순간 exclusive lock으로 lock의 속성이 변경된다.

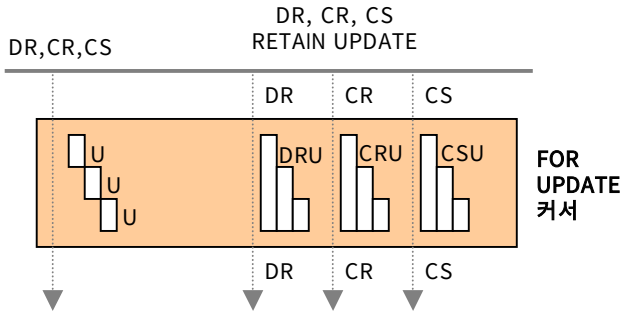
• 테이블이나 데이터 행을 접근하려 할 때 다른 세션이 그 오브젝트에 lock이 있는 상태라면 Lock 모드에 따라 다르게 동작된다.
  – **NOT WAIT** (default값) : 즉시 에러를 리턴한다
  – **WAIT 시간**(초) : lock이 해제될 때까지 지정한 시간 만큼 기다리고 그 이후에도 lock해제 안되면 에러를 리턴한다.
  – **WAIT** : lock이 해제될 때까지 무한정 기다린다

```
Informix> onstat –g sql
Sess  SQL            Current            Iso Lock       SQL  ISAM F.E.
Id    Stmt type      Database           Lvl Mode       ERR  ERR  Vers
251   SELECT         stores_demo        CR  Not Wait   0    0    9.03
249   SELECT         stores_demo        CR  Wait 10    0    0    9.03
247   -              stores _demo       CR  Wait 4     0    0    9.03
245   -              stores_demo        CR  Wait       0    0    9.03
```

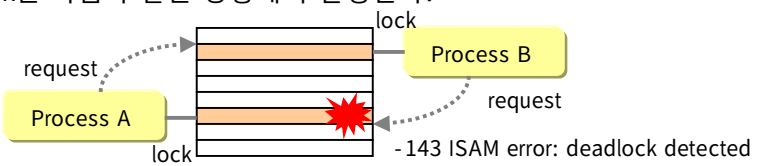
Locks and Concurrency

- **Retain Update Lock**
  - **SET ISOLATION TO DIRTY READ RETAIN UPDATE LOCKS;**
  - **SET ISOLATION TO COMMITTED READ RETAIN UPDATE LOCKS;**
  - **SET ISOLATION TO CURSOR STABILITY RETAIN UPDATE LOCKS;**
- **Deadlock 감지**
  - **deadlock 상황을 자동으로 감지하여 에러 메시지를 리턴한다.**

- Retain Update Lock은 SELECT ~ FOR UPDATE 커서를 사용하는 DIRTY READ, COMMITTED READ, CURSOR STABILITY 에서 적용 가능하다.
- 일반적인 FOR UPDATE 커서는 fetch하고 있는 데이터 행에 대해서만 update lock을 유지하지만, Retain Update Lock은 한번 update lock이 걸린 데이터 행은 FOR UPDATE 커서가 끝날 때까지 유지된다.



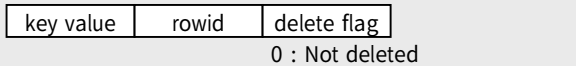
- Deadlock은 다음과 같은 상황에서 발생된다.



- 분산환경에서의 deadlock은 deadlock timeout 파라매터로 감지한다.

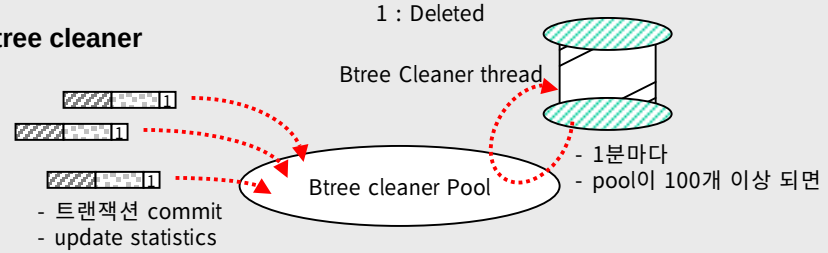
Locks and Concurrency

- **Key value locking**
  - **인덱스가 있는 데이터를 update,delete,insert 할 때 인덱스 Key에 locking하는 기법**



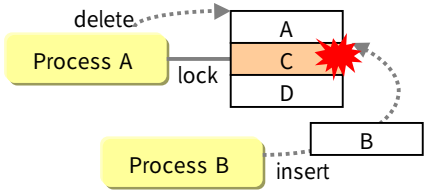
0 : Not deleted
1 : Deleted

- **Btree cleaner**

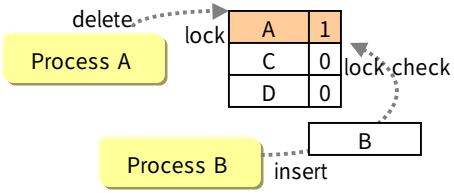


- **Shared Repeatable Read Lock**

- 버전6.0이후부터 인덱스 key를 delete할 때의 lock문제를 해결하기 위해 인덱스 key에 delete flag를 이용한다.



C에 lock을 걸고 insert 해야 하지만 A를 삭제하는 동안 C에 lock이 걸려 있어서 insert가 실패.

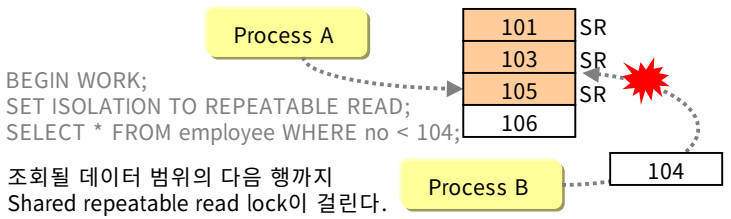


A를 삭제할 때 C에 lock을 걸지 않으므로 B를 insert할 때 lock 문제가 발생하지 않는다.

- delete flag가 1로 셋팅된 인덱스 키는 서버의 백그라운드 프로세스에 의해 나중에 삭제된다.
  - 데이터베이스가 로깅 모드가 아닌 경우 key value lock 없이 즉시 삭제.
  - 테이블의 exclusive lock이 있을 경우 데이터 및 인덱스키에 lock이 없음.

- Delete 를 실행시키면 물리적으로 인덱스 키를 삭제하는 것이 아니라 인덱스 delete flag만 1로 설정한다. 이런 인덱스 키는 트랜잭션이 commit되면 공유 메모리 안의 1Kb 크기의 btree cleaner pool에 들어간다. 이 pool이 100개가 넘거나 1분에 한 번씩 btree cleaner thread가 btree cleaner pool에 있는 인덱스 키를 삭제한다.
- 시스템 crash가 발생하면 메모리의 내용이 사라지므로 btree cleaner pool의 내용도 사라지게 된다. 이런 상황이면 인덱스 키가 delete flag만 셋팅 된 상태로 남아있게 되는데, UPDATE STATISTICS 구문을 실행시키면 이러한 인덱스 키를 찾아 btree cleaner pool에 넣게 된다. (UPDATE STATISTICS 구문은 다음 장에 설명된다)

- Shared Repeatable Read Lock



Exercise

- **Step 1 : Page lock**
- **Step 2 : Row lock**
- **Step 3 : Dirty Read**
- **Step 4 : Committed Read & lock mode wait**
- **Step 5 : Repeatable Read & lock mode not wait**

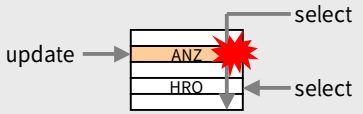
• Step 1 : page lock

```
<Session A>
BEGIN WORK;
UPDATE manufact SET lead_time = '2'
 WHERE manu_code = 'ANZ';
```


```
<Session B>


SELECT * FROM manufact
 WHERE manu_code = 'HRO';
# 243: Could not position ...
# 107: ISAM error: record is locked.
```

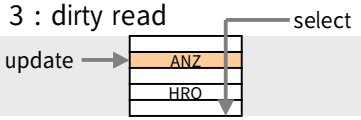
• Step 2 : row lock

```
ROLLBACK WORK;
ALTER TABLE manufact LOCK MODE (ROW);
BEGIN WORK;
UPDATE manufact SET lead_time = '2'
 WHERE manu_code = 'ANZ';
```



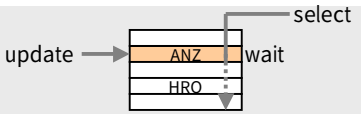

```
SELECT * FROM manufact
 WHERE manu_code = 'HRO';
SELECT * FROM manufact ;
#244: Could not do a physical-order read
#107: ISAM error: record is locked.
```

• Step 3 : dirty read



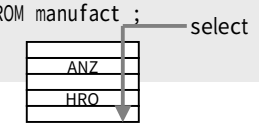

```
SET ISOLATION TO DIRTY READ;
SELECT * FROM manufact ;
```

• Step 4 : lock mode wait




```
COMMIT WORK;
```

```
SET ISOLATION TO COMMITTED READ;
SET LOCK MODE TO WAIT ;
SELECT * FROM manufact ;
```

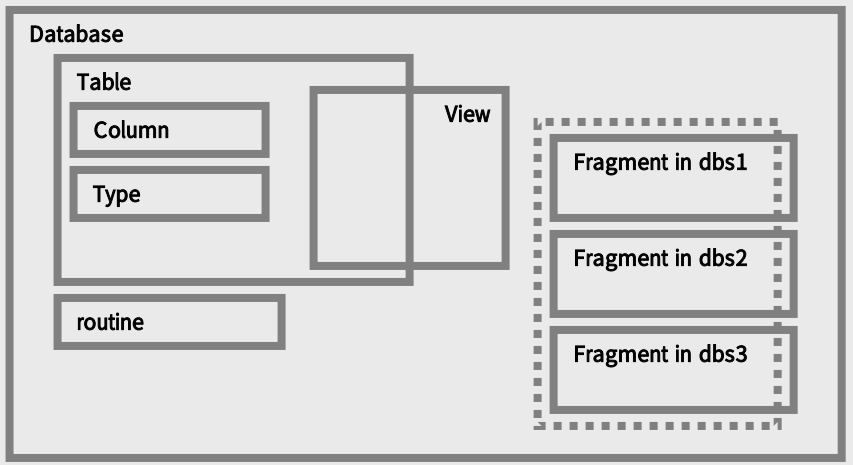


• Step 5 : lock mode not wait

```
SET ISOLATION TO REPEATABLE READ;
BEGIN WORK;
UPDATE manufact SET lead_time = '2'
 WHERE manu_name = 'Husky';
```


```
SET LOCK MODE TO NOT WAIT ;
UPDATE manufact SET lead_time = '2'
 WHERE manu_code = 'ANZ';
# 243: Could not position ...
# 113: ISAM error:  the file is locked.
SELECT * FROM manfuact
 WHERE manu_code = 'ANZ';
```

Levels of Data Security



- default 권한
  - Database 레벨
    - 데이터베이스를 생성하면 생성한 사람이 DBA권한을 갖는다
  - Table 레벨
    - Non-ANSI 데이터베이스에서 테이블을 생성하면 ALTER와 REFERENCE 권한을 제외한 ALL 권한이 부여된다.
    - ANSI 데이터베이스에서 테이블을 생성하면 default 권한이 없으므로 직접적으로 권한을 부여해야 한다.
    - Non-ANSI 데이터베이스에서 ANSI데이터베이스와 같이 default 테이블 권한을 부여하지 않으려면 테이블을 생성하기 전에 환경변수 NODEFDAC를 설정한다.

      export NODEFDAC=yes

- Database 레벨 권한
  - **CONNECT**
    - 데이터베이스에 접속할 수 있는 권한
    - 테이블, 인덱스를 생성 불가능
    - 임시테이블 및 임시테이블의 인덱스와 view, synonym 생성 가능
    - 자신이 만든 object에 대해 권한 부여 및 해지 가능
  - **RESOURCE**
    - CONNECT 권한 +
    - 테이블, 인덱스 생성/삭제 가능
    - Stored Procedure 및 function 생성/삭제 가능
  - **DBA**
    - RESOURCE 권한 +
    - 모든 레벨의 권한 부여 및 해지 가능
    - 데이터베이스 삭제 가능
    - 자기 자신의 DBA권한은 해지 불가능
- Table / Column 레벨 권한

테이블	권한	설 명	컬럼레벨
Read access	**SELECT**	데이터 조회	○
Modification access	**DELETE**	데이터 삭제	
	INSERT	데이터 입력	
	UPDATE	데이터 수정	○
Administration Access	**ALTER**	add,delete,modify 컬럼	
	INDEX	인덱스 생성	
	REFERENCE	컬럼 참조	○
전체	**ALL**	위의 모든 권한	

  - **UNDER** 권한은 9.x버전의 typed table에서만 적용된다(OR features)

Granting / Revoking the Privileges

- **Granting Database Level Privileges**
  - **GRANT DB_level_privileges TO { PUBLIC | user_list } ;**
- **Revoking Database Level Privileges**
  - **REVOKE DB_level_privileges FROM { PUBLIC | user_list } ;**
- **Granting Table Level Privileges**
  - **GRANT table_level_privileges ON table_name TO { PUBLIC | user_list } [WITH GRANT OPTION] [AS grantor] ;**
- **Revoking Table Level Privileges**
  - **REVOKE table_level_privileges ON table_name FROM { PUBLIC | user_list } [AS grantor] ;**

• PUBLIC은 모든 사용자를 나타내는 키워드, user_list는 콤마(,)로 구분한다.

```
GRANT CONNECT TO PUBLIC ;
GRANT RESOURCE TO maria, joe ;
GRANT DBA TO janet ;
```

• DBA나 RESOURCE권한을 revoke하더라도 CONNECT권한은 남아 있다. 따라서 데이터베이스 레벨의 모든 권한을 해지하고 싶으면 CONNECT권한에 대해서도 revoke 해야 한다.

```
REVOKE CONNECT FROM mike ;
REVOKE RESOURCE FROM maria ;
```

• 테이블 레벨의 권한은 뷰(view)에 대해서도 적용 가능하다
• WITH GRANT OPTION을 사용하면 테이블 레벨의 권한을 부여하면서 그 권한을 다른 사람에게 grant할 수 있는 권한까지 넘겨 주게 된다.
• 권한을 부여하는 사람을 AS를 사용하여 지정할 수 있다.

```
GRANT ALL ON customer TO PUBLIC ;
GRANT UPDATE ON orders TO liz WITH GRANT OPTION ;
GRANT INSERT, DELETE ON items TO mike AS maria ;
```

현재 구문을 실행하는 사람이 아니라 maria가 mike에게 권한을 부여하는 것으로 인식된다

liz가 다른 사용자에게 update 권한을 부여할 수 있게 된다 만약, liz가 update 권한을 잃게 되면 liz에게 update권한을 부여 받은 다른 사용자도 함께 권한을 잃게 된다

• 컬럼 단위로 grant되었더라도 revoke는 테이블단위로 수행해야 한다.
• REOVKE ~ AS 구문은 9.3버전부터 사용할 수 있다.

```
REVOKE ALL ON orders FROM PUBLIC ;
REVOKE DELETE, UPDATE ON customer FROM mike, maria;
REVOKE INSERT, UPDATE ON items FROM mike AS maria;
```

Granting / Revoking the Privileges

- **Granting Column Level Privileges**
  - **GRANT column_level_privileges (column_name_list)**
    ON table_name TO { PUBLIC | user_list } ;
- **Granting UNDER Privileges**
  - **GRANT UNDER ON { TYPE named_row_type | table_name }**
    TO { PUBLIC | user_list }
    [WITH GRANT OPTION] [AS grantor] ;
- **Revoking UNDER Privileges**
  REVOKE UNDER ON { TYPE named_row_type | table_name }
    FROM { PUBLIC | user_list } [CASCADE | RESTRICT] ;

• 컬럼 단위로 부여 가능한 권한은 SELECT, UPDATE, REFERENCES이다
• 컬럼 단위로 부여 받은 권한을 컬럼 단위로 해지할 수 없으므로 테이블 단위로 revoke해야 한다.

```
GRANT SELECT (company, fname, lname) ON customer TO PUBLIC ;
GRANT INSERT, UPDATE (quantity), SELECT ON items TO maria ;
```

• GRANT UNDER 예제

```
<user1>
CREATE ROW TYPE type1
(x INT, y INT);
CREATE ROW TYPE type2
(a INT, b INT) UNDER type1;


GRANT UNDER ON TYPE type1 TO user2 ;

CREATE TABLE typedtable1 OF TYPE type1;
CREATE TABLE typedtable2 OF TYPE type2
 UNDER typedtable1 ;

GRANT UNDER ON typedtable1 TO user2;
```

```
<user2>


CREATE ROW TYPE type3
(i INT, j INT) UNDER type1;
# 9653: UNDER privilege required to
        create subtype/subtable.
CREATE ROW TYPE type3
(i INT, j INT) UNDER type1; -- OK

CREATE TABLE typedtable2 OF TYPE type2
 UNDER typedtable1 ;
# 9653: UNDER privilege required to …
CREATE TABLE typedtable2 OF TYPE type2
 UNDER typedtable1 ;-- OK
```

• UNDER 권한을 revoke할 때 CASCADE 옵션을 사용하면 typed table로 부터 파생(derived)된 subtable도 삭제된다.
• RESTRICT 옵션은 해당 type이나 typed table이 다른 subtable로 파생시켰다면 이 type이나 typed table은 사용중인 상태(in use)로 REVOKE 구문이 실패하게 한다.

```
REVOKE UNDER ON TYPE type1 FROM PUBLIC RESTRICT ;
REVOKE UNDER ON type_table FROM carol CASCADE ;
```

Granting / Revoking the Privileges

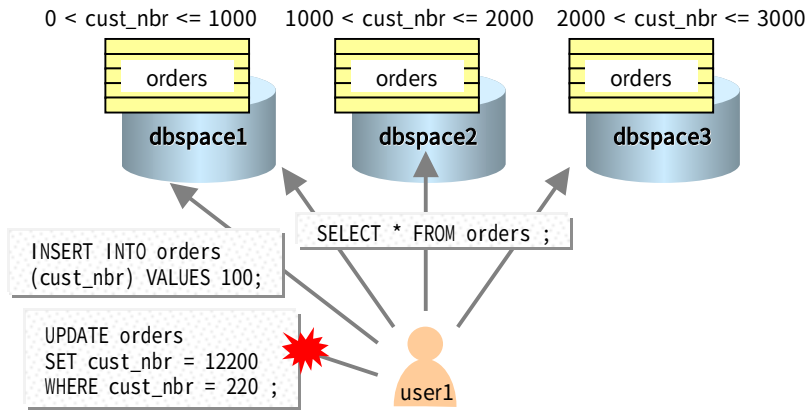
- **Routine Privileges**
  - **GRANT EXECUTE ON routine_name TO { PUBLIC | user_list } ;**
  - **REVOKE EXECUTE ON routine_name FROM { PUBLIC | user_list } ;**
- **Fragment Privileges**
  - **GRANT FRAGMENT fragment_privileges ON table_name (dbspace_list) TO { PUBLIC | user_list } [WITH GRANT OPTION] [AS grantor] ;**
  - **REVOKE FRAGMENT fragment_privileges ON table_name [(dbspace_list)] FROM { PUBLIC | user_list } [AS grantor] ;**

• Routine 권한 예제

```
GRANT EXECUTE ON total_orders TO PUBLIC ;
GRANT EXECUTE ON square (x INT) TO maria ;
REVOKE EXECUTE ON cancel_orders FROM joe, tom ;
```

• Fragment 권한은 INSERT, UPDATE, DELETE 이다.
• Fragment 권한은 expression 방식의 fragment에서만 적용 가능하고, 테이블 단위의 권한이 더 우선한다. 예를 들어 테이블에 INSERT 권한이 있다면 fragment 권한은 체크하지 않고 모든 fragment에 insert할 수 있다

```
REVOKE ALL ON orders FROM PUBLIC;
GRANT SELECT ON orders TO PUBLIC;

REVOKE FRAGMENT ALL ON orders FROM user1;
GRANT FRAGMENT INSERT,UPDATE,DELETE ON orders(dbspace1) TO user1;
```



Using Roles

- **Role 생성**
  - **CREATE ROLE role_name ;**
- **사용자를 Role에 등록**
  - **GRANT role_name TO user_list ;**
- **Role에 권한 부여**
  - **GRANT table_level_privileges ON table_name TO role_name ;**
- **Role 사용**
  - **SET ROLE { role_name | NULL | NONE } ;**

- Role은 동일한 권한을 가지는 user들의 모임이라 할 수 있다.
- Role 이름은 8자 이내로 만들어야 하고 데이터베이스에 연결할 수 있는 user 명, 즉 /etc/passwd파일에 등록된 id를 사용할 수 없다.
- CREATE ROLE구문은 해당 데이터베이스에 DBA권한을 가지고 있어야 실행 가능하다.

```
CREATE ROLE root;
# 19800: Role name already exists as a user or role.
CREATE ROLE mkting;
CREATE ROLE slsadmin;
CREATE ROLE sales;
```

```
GRANT mkting TO jim, mary, ram;
GRANT slsadmin TO andy, liz, sam;
GRANT sales TO mkting, slsadmin;
```

```
REVOKE ALL ON orders FROM public;
GRANT SELECT ON orders TO sales;
GRANT INSERT, UPDATE, DELETE ON orders TO slsadmin;
```

- Role은 세션이 끝나기 전이나 SET ROLE NULL 또는 SET ROLE NONE 구문을 실행하기 전까지 적용 받는다.

```
<liz>
INSERT INTO ORDERS
       (order_num,customer_num)
       VALUES (0,104);
# 275: No INSERT permission.
SET ROLE slsadmin ;
INSERT INTO ORDERS
       (order_num,customer_num)
       VALUES (0,104);
SET ROLE NULL ;
```

```
<mary>



SET ROLE slsadmin ;
# 111: ISAM error:  no record found.
SET ROLE mkting ;
SELECT * FROM orders;
# 272: No SELECT permission.
```

Views

- 뷰(view)는 물리적으로 저장된 테이블이 아닌 가상테이블(virtual table)로 실제 존재하는 테이블로부터 데이터를 가져온다.
- 또한 저장된 값이 아닌 계산된 값을 데이터 컬럼과 같이 사용할 수 있으므로 dynamic window라 부르기도 한다.

- 뷰(view)는 보안의 목적으로 원본 테이블의 일부분만을 사용할 수 있다
  - 일부 컬럼만 접근
  - 일부 행만 접근
- 뷰(view)는 단순한 SQL사용을 위한 목적으로 구현될 수 있다
  - 가상 컬럼(virtual column)
  - 집계 함수(aggregate function)
  - 테이블 조인
- 뷰(view)는 다른 뷰(view)에서 사용 될 수 있다

• View 생성 및 삭제 예제

```
CREATE VIEW ordsummary AS SELECT order_num, customer_num, ship_date FROM orders ;
CREATE VIEW ordnotpaid (ordno, orddate, cnum)
    AS SELECT order_num, corder_date, customer_num FROM orders
        WHERE paid_date IS NULL ;
DROP VIEW ordsummary ;
```

select list의 컬럼명이 그대로 view의 컬럼이 된다.

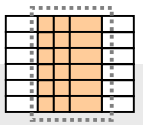
view는 alter할 수 없으므로 먼저 drop하고 새로 create해야 한다.

view의 컬럼명을 지정하고 원본 테이블에서 조건절을 지정하였다

• View를 생성할 때 사용하는 select에는 다음과 같은 구문을 사용할 수 없다.
  – FIRST
  – ORDER BY
  – INTO TEMP

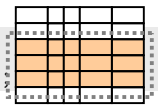
• Views : Access to columns

```
CREATE VIEW ordsummary
    AS SELECT order_num, customer_num, ship_date FROM orders ;
```



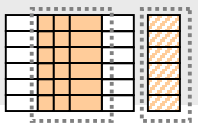
• Views : Access to Rows

```
CREATE VIEW baseball
    AS SELECT * FROM stock WHERE description MATCHES “*baseball*”
```



• Views : A Virtual Column

```
CREATE VIEW ship_cost (ordno, cnum, s_wt, s_chg, chg_per_lb)
    AS SELECT ord_num, customer_num, ship_weight, ship_charge,
              ship_charge / ship_weight
         FROM orders ;
```



• Views : An Aggregate Function

```
CREATE VIEW manu_total (m_code, total_sold)
    AS SELECT manu_code, SUM(total_price)
         FROM items
        GROUP BY manu_code ;
```

• Views : A view that Joins Two Tables

```
CREATE VIEW stock_info
    AS SELECT stock.*, manu_name
         FROM stock, manufact
        WHERE stock.manu_code = manufact.manu_code ;
```

• Views : A view on another view

```
CREATE VIEW manu_new
    AS SELECT manu_name, total_sold
         FROM manufact, manu_total
        WHERE manufact.manu_code = manu_total.m_code ;
```

3

Views

- **제한사항**
  - **뷰(view)에 인덱스를 생성할 수 없다**
  - **뷰의 옵션에 따라 insert, update, delete의 제한이 있을 수 있다**
  - **뷰를 생성하기 위해서는 기본 테이블에 select 권한이 있어야 한다**
  - **집계함수나 join을 사용한 뷰는 insert, update, delete가 불가능하다**
  - **가상컬럼을 사용한 뷰는 insert, update가 불가능하다**
- **WITH CHECK OPTION**
  - **뷰의 생성 범위를 벗어난 데이터를 입력할 수 없도록 한다**
- **뷰에 대한 권한은 다음과 같은 경우를 제외하고는 일반 테이블에 대한 권한과 동일하게 사용할 수 있다**
  - **ALTER 권한은 없음**
  - **뷰에 대한 제한사항이 있으면 INSERT, UPDATE 등의 권한 부여가 불가능하다**

- 뷰 자체에는 인덱스를 생성할 수 없지만 select를 하면 기본 테이블에 존재하는 인덱스를 사용하게 된다.
- 뷰의 기본 테이블을 drop하면 뷰도 함께 drop된다.

- WITH CHECK OPTION

```
CREATE VIEW no_check
    AS SELECT * FROM stock WHERE manu_code = "HRO";
CREATE VIEW yes_check
    AS SELECT * FROM stock WHERE manu_code = "HRO"
    WITH CHECK OPTION ;
INSERT INTO no_check VALUES (1, "ANZ", "soccer ball", 30, "each", "each");
INSERT INTO yes_check VALUES (1, "ANZ", "soccer ball", 30, "each", "each");
# 385: Data value out of range.
```

  - no_check 뷰의 경우 어떠한 데이터도 기본테이블 stock에 입력되지만 no_check를 통해 그 데이터를 조회할 수 없다.
  - yes_check 뷰는 생성할 때의 조건과 맞는 데이터만 입력할 수 있으므로 데이터 테이블에 대한 보안성을 좀 더 강화할 수 있다.

- 다음과 같이 실행하면 기본테이블 stock에 대해 접근할 수 있는 방법은 뷰 stock_info 를 접근하는 방법 밖에는 없다

```
REVOKE ALL ON stock FROM PUBLIC ;
REVOKE ALL ON stock_info FROM PUBLIC ;
GRANT SELECT ON stock_info TO dennis, karen, mari ;
```

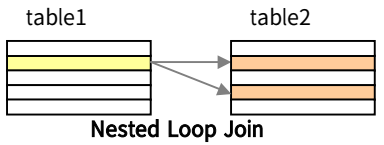
4. Optimizing the Databases

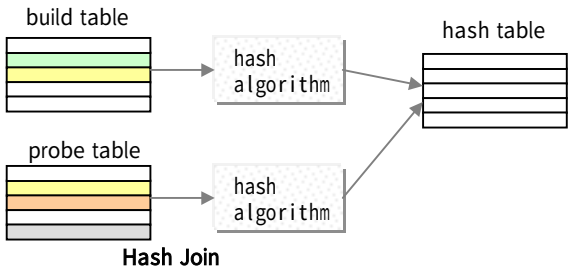
The Cost-based Query Optimizer
Viewing the Query Plan
Optimizing Subqueries
Influencing the Optimizer / OPTCOMPIND
SET OPTIMIZATION
USING DIRECTIVES
The SQL Statement Cache (SSC)
Update Statistics
Distributions
Resolution / Confidence
Update Statistics Guideline

The Cost-based Query Optimizer

- **Query Plan**
  - **Optimizer가 선택하는 access plan과 join plan으로 결정**
  - **조회할 페이지를 줄이고 불필요한 정렬 작업을 최소화하도록 선택**
- **Access Plan 작성 단계**
  - **각 필터를 통해 예상되는 데이터 출력 건수 산정 (selectivity)**
  - **인덱스 사용 가능 여부 검사**
    - **filter 컬럼**
    - **ORDER BY / GROUP BY 컬럼**
  - **최적의 access 방안 선택**
    - **sequentially**
    - **by an index**
- **Join Plan 작성 단계**
  - **테이블간의 join 조합 구성**
  - **각 조합별로 I/O 및 CPU cost 산정**
  - **부하가 많은 (cost가 크다. expensive) 조합은 plan에서 제거**

- Access Plan
  - **Sequential scan** : 물리적 저장 순서로 모든 데이터 페이지 조회
  - **Index scan** : 인덱스 페이지를 먼저 조회하고 데이터 페이지 조회
  - **Key-only index scan** : 조회할 데이터가 인덱스 컬럼만으로 구성되어 있을 때 인덱스 페이지만 조회
  - **Key-first index scan** : 조회할 데이터 행 수를 줄이기 위하여 filter 역할로 인덱스 페이지를 먼저 사용
  - **Auto-index scan** : query filter로서 사용될 컬럼에 자동적으로 임시적인 인덱스를 먼저 생성하고 조회
- Join Plan
  - **nested loop join** : 첫번째 테이블의 데이터를 읽고 두번째 테이블에서 관련된 행을 찾는다. 주로 두번째 테이블에서는 인덱스를 사용하게 된다
  - **hash join** : 첫번째 테이블(build table)에서 hash 알고리즘으로 hash row를 추출하고 관련된 행을 두번째 테이블(probe table)에서 찾아 hash 알고리즘을 통해 hash 테이블을 생성하고 이 hash 테이블을 통해 데이터를 조회한다.





Viewing the Query Plan

- **Query plan 확인**
  - **SET EXPLAIN { ON | OFF } ;**
- **EXPLAIN을 ON시키고 SQL구문을 실행하면 query plan을 확인 할 수 있는 텍스트 파일이 생성된다.**
  - **예상 비용 (estimate cost)**
  - **테이블 접근 순서**
  - **임시테이블 사용 정보**
  - **access method / join method**
- **Explain 파일**
  - **UNIX : ${PWD}/sqexplain.out 또는 ${HOME}/sqexplain.out**
  - **NT : %INFORMIXDIR%\sqexpln*username*.out**
- **Example**
  - **Sequential Scan with Temporary Table**
  - **Sequential Scan with Filter**

- Explain을 ON시키면 세션을 종료하거나 explain을 off시키기 전까지 모든 SQL구문에 대해 query plan을 explain 파일에 기록한다. 그러므로 사용자 프로그램에 explain을 on시키는 구문을 넣지 않도록 주의한다.
- Query plan은 explain파일에 계속 append 된다.
- Explain 파일은 UNIX에서 sqexplain.out으로 생성되는데, 보통 현재 SQL을 실행하는 디렉토리에 생성되고, remote system에서 접속한 경우 그 사용자의 홈 디렉토리에 생성된다.
- Explain 파일은 query가 실행되기 전에 작성되는데, 9.3버전 부터는 query 를 실행하지 않고도 explain 파일을 생성할 수 있다.
  - – SET EXPLAIN ON AVOID_EXECUTE ;

- Sequential Scan with Temporary Table

```
QUERY:
------
select * from stock order by description

Estimated Cost: 19
Estimated # of Rows Returned: 74
Temporary Files Required For: Order By

  1) informix.stock: SEQUENTIAL SCAN
```

cost의 단위는 없으며 절대적인 수치가 아니므로 동일한 SQL의 query path를 비교하는데 사용한다

optimizer가 예상하는 수이므로 실제와 차이가 있다. 테이블의 catalog정보가 최근 갱신되었다면 좀 더 정확한 수치가 출력된다

order by에 의해서 임시 파일이 필요함을 나타낸다. 인덱스 컬럼을 통해 order by를 수행했다면 임시 파일이 필요 없다

access method

- Sequential Scan with Filter

```
QUERY:
------
select * from stock where unit_price > 20

Estimated Cost: 4
Estimated # of Rows Returned: 25

  1) informix.stock: SEQUENTIAL SCAN

        Filters: informix.stock.unit_price > $20.00
```

sequential scan을 하면서 조건에 맞는 행을 걸러 준다

Viewing the Query Plan

- **Example (계속)**
  - **Key-only Index Scan**
  - **Index Scan with Lower Index Filter**
  - **Index Scan with Lower and Upper Index Filters**

• Key-only Index Scan

```
QUERY:
------
select max(order_num) from orders

Estimated Cost: 1
Estimated # of Rows Returned: 1

  1) informix.orders: INDEX PATH

    (1) Index Keys: order_num  (Key-Only)  (Aggregate)  (Serial, fragments: ALL)
```

인덱스가 생성된 컬럼을 사용한 query 이므로 가장 빠른 query path라 예상할 수 있다

인덱스 컬럼 데이터만 스캔하면 조회 결과를 얻을 수 있으므로 데이터 페이지 스캔이 일어나지 않는다

• Index Scan with Lower Index Filter

```
QUERY:
------
select * from stock, items
where stock.stock_num = items.stock_num
and items.quantity > 1
Estimated Cost: 11
Estimated # of Rows Returned: 39

  1) informix.items: SEQUENTIAL SCAN
        Filters: informix.items.quantity > 1

  2) informix.stock: INDEX PATH
    (1) Index Keys: stock_num manu_code   (Serial, fragments: ALL)
        Lower Index Filter: informix.stock.stock_num = informix.items.stock_num

NESTED LOOP JOIN
```

인덱스를 통하여 여러 건의 데이터를 조회할 때, 인덱스로 처음 찾은 데이터부터 시작하여 sequential하게 조회한다. 이때 인덱스를 통하여 처음으로 찾는 데이터를 지정하는 조건이 lower index filter가 된다

items 테이블의 각 행마다 조인 컬럼을 이용하여 stock테이블에서 index 스캔을 한다

• Index Scan with Lower and Upper Index Filters

```
QUERY:
------
select * from customer
where customer_num between 104 and 111

Estimated Cost: 2
Estimated # of Rows Returned: 8

  1) informix.customer: INDEX PATH

    (1) Index Keys: customer_num   (Serial, fragments: ALL)
        Lower Index Filter: informix.customer.customer_num >= 104
        Upper Index Filter: informix.customer.customer_num <= 111
```

인덱스 페이지 조회를 중단하는 지점을 나타내는 조건을 upper index filter라 한다

Viewing the Query Plan

- **Example (계속)**
  - **Dynamic Hash Join**
  - **Key-First Index Scan**

• Dynamic Hash Join

```
QUERY:
------
select * from items, stock
where items.total_price = stock.unit_price

Estimated Cost: 33
Estimated # of Rows Returned: 496

  1) informix.items: SEQUENTIAL SCAN
  2) informix.stock: SEQUENTIAL SCAN (Parallel, fragments: ALL)

DYNAMIC HASH JOIN
    Dynamic Hash Filters: informix.items.total_price = informix.stock.unit_price
```

hash 테이블을 자동적으로 생성하여 hash join을 수행한다. build outer 문장이 표시 되지 않았다면 기본적으로 두번째 테이블에 대하여 hash 테이블이 생성된다.

sequential scan이 병렬로 동작된다

– 일반적으로 DSS환경에서는 테이블을 full scan하여 대량의 데이터를 읽는 경우가 많다. 이러한 환경, 특히 join 결과가 매우 큰 경우 hash join은 다른 join기법에 비해 월등한 성능을 나타낸다.
– 만약 테이블이 fragment 되고 PDQ(Parallel Data Query) 를 설정하면, sequential scan이 병렬로 동작된다

• Key-First Index Scan

```
QUERY:
------
select * from customer
where (customer_num > 120)
and (customer_num = 122 or customer_num = 123)

Estimated Cost: 2
Estimated # of Rows Returned: 1

  1) informix.customer: INDEX PATH

    (1) Index Keys: customer_num  (Key-First) (Serial, fragments: ALL)
        Lower Index Filter: informix.customer.customer_num > 120
        Key-First Filters: ((informix.customer.customer_num = 122
                         OR informix.customer.customer_num = 123 ) )
```

데이터 조회량을 줄이기 위하여 index를 filter로 사용함으로써 불필요한 데이터를 제거한다

Optimizing Subqueries

- **Subquery Flattening**
  - **Optimizer가 자동적으로 subquery를 join형태로 바꾸어 최적화**
- **Subquery flattening을 위한 special access plan**
  - **Skip duplicate index scan**
    - 중복된 인덱스 데이터가 있어도 하나만 리턴
  - **First row scan**
    - 조회하는 데이터 중 처음 일치하는 행을 만나면 그 행만 리턴
  - **Semi-join**
    - nested loop join을 할 때 inner 테이블에서 처음으로 일치하는 행을 만날 때 까지만 스캔

- subquery는 SELECT, INSERT, DELETE, UPDATE 문장에서 WHERE절에 포함된 select절을 의미한다. 특히 correlated subquery는 outer 테이블의 각 행마다 subquery를 실행시켜야 하므로 성능이 떨어진다.

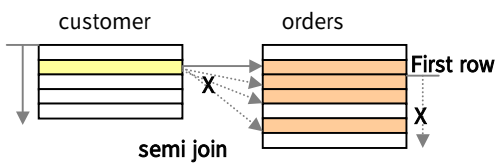
```
-- subquery
SELECT * FROM customer WHERE customer_num IN
       (SELECT customer_num FROM orders WHERE order_date = TODAY );
-- correlated subquery
SELECT * FROM customer c WHERE EXISTS
       (SELECT customer_num FROM orders
         WHERE orders.customer_num = c.customer_num AND order_date = TODAY );
```

- 보통 correlated subquery는 join형태로 구현하는 것이 바람직하다

```
SELECT c.* FROM customer c, orders o
 WHERE o.customer_num = c.customer_num AND order_date = TODAY ;
```

- First Row / Semi join

```
QUERY:
------
select customer_num, fname, lname from customer
where exists
(select customer_num from orders
  where customer.customer_num = orders.customer_num)

Estimated Cost: 8
Estimated # of Rows Returned: 22

  1) informix.customer: SEQUENTIAL SCAN

  2) informix.orders: INDEX PATH (First Row)

    (1) Index Keys: customer_num   (Key-Only)  (Serial, fragments: ALL)
        Lower Index Filter: informix.customer.customer_num
                          = informix.orders.customer_num
NESTED LOOP JOIN (Semi Join)
```



Optimizing Subqueries

- **Predicate Promotion Access Control Blocks**
  - **Correlated subquery 중 non-correlated subquery로 재작성 할 수 있는 경우 optimizer가 자동적으로 변환된 형태로 query plan 결정**

• 다음과 같은 correlated subquery는 아래와 같이 non-correlated subquery로 재작성 할 수 있다.

```
-- correlated subqeury
SELECT * FROM customer c
 WHERE state = "CA"
   AND EXISTS ( SELECT "X" FROM state s WHERE c.state = s.code) ;
-- non-correlated subquery
SELECT * FROM customer c
 WHERE state = "CA"
   AND EXISTS ( SELECT "X" FROM state s WHERE s.code = "CA") ;
```

• 위와 같은 경우 optimizer는 자동적으로 동일한 query plan을 결정한다.

```
QUERY:
------
SELECT * FROM customer c
WHERE state = "CA"
AND EXISTS ( SELECT "X" FROM state s WHERE c.state = s.code)

Estimated Cost: 2
Estimated # of Rows Returned: 3

  1) informix.c: SEQUENTIAL SCAN

        Filters: (informix.c.state = 'CA' AND EXISTS <subquery> )

    Subquery:
    ---------
    Estimated Cost: 1
    Estimated # of Rows Returned: 5

      1) informix.s: INDEX PATH

        (1) Index Keys: code   (Key-Only)  (Serial, fragments: ALL)
            Lower Index Filter: informix.s.code = 'CA'
```

Influencing the Optimizer / OPTCOMPIND

- **OPTCOMPIND**
  - **Access path (index, sequential scan)을 결정하는 config 파라매터 또는 환경변수**
- **SET OPTIMIZATION 구문**
  - **Query path를 결정할 때 최적화 목표(goal)를 지정**
- **OPTIMIZER DIRECTIVES**
  - **SQL구문 안에서 다양한 최적화 방안을 명시적으로 지정**

• OPTCOMPIND는 config 파라매터로 ($INFORMIXDIR/etc/$ONCONFIG) 모든 사용자 세션에 global하게 적용되는데, 각 세션에서 환경변수로 지정하면 환경변수 값이 우선적으로 적용된다.

OPTCOMPIND	설 명
0	Access Path : 인덱스가 있는 경우 Index scan Join Plan : nested loop join
1	Isolation이 repeatable read이면 OPTCOMPIND=0와 동일 그외의 isolation이면 OPTCOMPIND=2와 동일
2 (default)	Optimizer가 최소 비용의 query plan결정

• OPTCOMPIND=2 일 때의 query plan의 예

```
QUERY:
------
select count(*) from customer where customer_num between 100 and 105

Estimated Cost: 1
Estimated # of Rows Returned: 1
  1) informix.customer: INDEX PATH

    (1) Index Keys: customer_num  (Key-Only)  (Serial, fragments: ALL)
        Lower Index Filter: informix.customer.customer_num >= 100
        Upper Index Filter: informix.customer.customer_num <= 105


QUERY:
------
select count(*) from customer where customer_num between 100 and 200

Estimated Cost: 2
Estimated # of Rows Returned: 1

  1) informix.customer: SEQUENTIAL SCAN

        Filters: (informix.customer.customer_num <= 200
              AND informix.customer.customer_num >= 100 )
```

동일한 query이지만 데이터 분포 상 index scan / sequential scan 을 판단

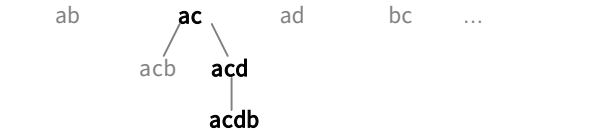
4

SET OPTIMIZATION

- **Default값은 HIGH와 ALL_ROWS**
- **적용된 내용은 세션이 끝나거나 다음 SET OPTIMIZATION 구문을 실행하기 전까지 유효**
- **한 문장에 하나의 옵션만 적용**
  - **SET OPTIMIZATION { HIGH | LOW } ;**
  - **SET OPTIMIZATION { FIRST_ROWS | ALL_ROWS } ;**

- FIRST_ROWS vs. ALL_ROWS
  - **FIRST_ROWS** : 조건을 만족하는 데이터 셋의 첫번째 행이 가장 빠르게 조회 될 수 있도록 최적화 함. 보통 end-user query 툴을 사용할 때 처음 조회된 결과값으로 다음 query를 수행하는 경우 바람직한 방법이다.
  - **ALL_ROWS** : 전체 데이터를 조회 결과로 리턴할 때의 최적화 방안으로 대부분의 경우 가장 많이 적용된다.
- HIGH vs. LOW
  - **HIGH** : query path를 결정하는데 가장 정확한 옵션으로 모든 가능한 query path에 대한 조합을 만들어 가장 적은 cost를 결정한다.
  - **LOW** : 단계별로 cost가 적은 query path만 선택하는 형태로 optimizing 결과는 HIGH에 비해 덜 정확할 수 있으나 optimizing 수행 시간은 단축시킬 수 있다

```
SET OPTIMIZATION LOW ;
SELECT * FROM a,b,c,d
WHERE a.a = b.a AND b.b = c.b AND c.c = d.c ;
```



- 다음과 같은 경우 OPTIMIZATION을 LOW로 설정한다.
  - Join 테이블이 4~5개 이상 일 경우
  - 모든 join 컬럼에 인덱스가 생성된 경우
  - 하나의 테이블이 여러 테이블과 join된 경우
  - 전체 query 수행시간 중 optimizing 시간만을 측정할 수 없으므로 전체 query 수행시간이 만족스럽지 않은 경우 성능 비교
- OPTIMIZATION을 FIRST_ROWS 로 설정하기 위한 방법
  - SQL 구문 : SET OPTIMIZATION FIRST_ROWS ;
  - Config 파라매터 : OPT_GOAL 0 (-1은 ALL_ROWS)
  - 환경변수 : export OPT_GOAL=0
  - SQL optimizer directives (directives는 다음 페이지에 설명된다)

```
SELECT --+ FIRST_ROWS
        fname, lname
  FROM customer
 ORDER BY lname ;
```

- 적용우선 순위 :
  - SQL Optimizer directives → SQL구문 → 환경변수 → config 파라매터

USING DIRECTIVES

- **SQL에 직접적으로 optimizing 방법을 명시하는 구문을 optimizer directives 라고 한다**
  - **Positive Directives**
  - **Negative Directives**

- **Optimizer Directives의 종류**
  - **Access method directives**
  - **Join order directives**
  - **Join method directives**
  - **Optimization goal directives**
  - **Explain diretivies**

- **SQL comment 문자 뒤에 + 기호를 붙여서 directives를 나타낸다**
  - **--+ directives 구문**
  - **{+ directives 구문 }**
  - **/*+ directives 구문 */**

- 특정 query path를 지정하는 positive directives 외에 피해야 할 (avoid) query path를 지정하는 negative directives 개념은 다른 DBMS와 차별되는 특징이다

- Optimizer directives의 종류는 다음과 같다
  - Access method : INDEX, AVOID_INDEX, FULL, AVOID_FULL
  - Join order
  - Join method : USE_NL, AVOID_NL, USE_HASH, AVOID_HASH
  - Optimization goal : FIRST_ROWS, ALL_ROWS
  - Explain

- Optimizer directives는 config 파라매터 ($INFORMIXDIR/etc/$ONCONFIG) DIRECTIVES값이 1(ON, default)로 설정되어 있을 때 사용할 수 있다. 만약 DIRECTIVES가 0(OFF)이더라도 환경변수 IFX_DIRECTIVES를 ON 또는 OFF 시켜 세션 단위로 제어할 수 있다.
  - export IFX_DIRECTIVES=ON

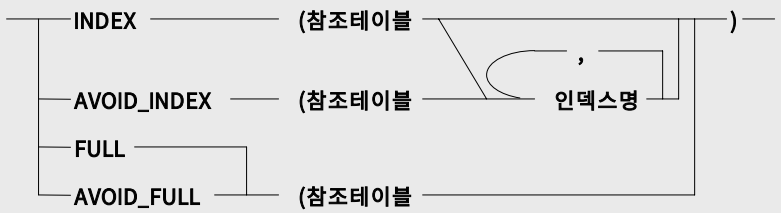
- Optimizer directives는 거의 모든 SQL에서 사용 가능하다
  - SELECT, UPDATE, DELETE 구문
  - INSERT문에 포함된 SELECT문
  - View, Stored Procedures, Triggers
- 하지만 다음과 같은 경우는 optimizer directives를 사용할 수 없다
  - 분산 DB의 테이블에 대해서
  - WHERE CURRENT OF구문을 사용한 UPDATE, DELETE 구문

- Optimizer directives는 comment 문자열 뒤에 +기호를 붙여 나타내고 그 comment block안에서 여러 개의 optimizer directives를 사용할 수 있다

- Optimizer directives 구문이 정확하지 않을 경우에도 error를 출력하지 않고 정상적으로 SQL을 실행한다. 이때 explain 출력 결과를 보면 다음 문장 아래 directives에 대한 내용이 없다
  DIRECTIVES FOLLOWED
  DIRECTIVES NOT FOLLOWED

USING DIRECTIVES

- **Access Method Directives**
  - **INDEX**
  - **AVOID_INDEX**
  - **FULL**
  - **AVOID_FULL**



※ 참조테이블 : table, synonym, alias

• INDEX
  – 테이블에 인덱스가 있다면 sequential scan을 하지 않는다.
  – Directives에 두개 이상의 인덱스를 지정하면 optimizer가 cost를 비교하여 사용할 인덱스를 결정하고 인덱스 이름을 지정하지 않으면 그 테이블에 생성된 모든 인덱스에 대하여 cost를 비교한다.

```
SELECT --+ INDEX (e salary_indx)
       name, salary
  FROM emp e
 WHERE e.dno = 1
   AND e.salary > 50000 ;
```

• AVOID_INDEX
  – 지정한 인덱스를 사용하지 않도록 한다.

```
SELECT --+ AVOID_INDEX (e salary_indx)
       name, salary
  FROM emp e
 WHERE e.dno = 1
   AND e.salary > 50000 ;
```

• FULL
  – 인덱스가 있는 테이블이라 하더라도 sequential scan을 하도록 한다

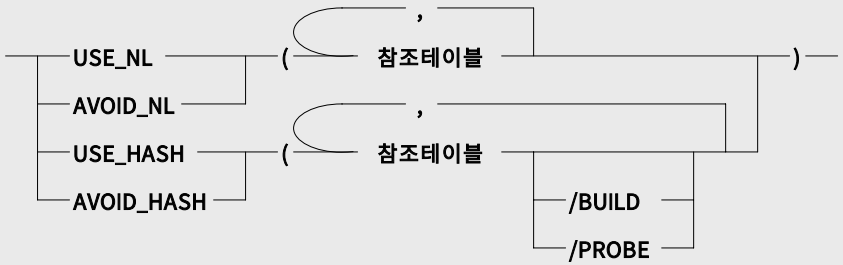
```
SELECT --+ FULL (e)
       name, salary
  FROM emp e ;
```

• AVOID_FULL
  – 인덱스가 존재한다면 sequential scan을 하지 않도록 한다

```
SELECT --+ AVOID_FULL (e) , INDEX(e salary_indx)
       name, salary
  FROM emp e
 WHERE e.dno = 1
   AND e.salary > 50000 ;
```

USING DIRECTIVES

- **Join Method Directives**
  - **USE_NL**
  - **AVOID_NL**
  - **USE_HASH**
  - **AVOID_HASH**



- USE_NL
  - 지정한 테이블을 inner table로서 nested loop join을 수행한다.
  - Nested loop join은 outer table의 각 row에 대하여 inner table의 일치하는 행을 찾는데, inner table은 기존에 존재하거나 자동적으로 생성된 인덱스를 가지고 스캔한다.

```
SELECT --+ USE_NL (dept)
       name, title, salary
  FROM emp, dept, job
 WHERE loc = “Palo Alto”
   AND emp.dno = dept.dno
   AND emp.job = job.job ;
```

- AVOID_NL
  - 지정한 테이블이 nested loop join을 수행하지 않도록 한다

- USE_HASH
  - Hash join을 수행하도록 지정할 수 있는데, 명시적으로 probe, build 테이블을 정의하지 않으면 optimizer가 cost를 비교하여 결정한다

```
SELECT --+ USE_HASH (dept /BUILD)
       name, title, salary
  FROM emp, dept, job
 WHERE loc = “Palo Alto”
   AND emp.dno = dept.dno
   AND emp.job = job.job ;
```

- AVOID_HASH
  - 지정한 테이블이 hash join을 수행하지 않도록 한다

USING DIRECTIVES

- **Join Order Directives**
  - **ORDERED**
- **Optimization Goal Directives**
  - **FIRST_ROWS**
  - **ALL_ROWS (default)**
- **Explain Directives**
  - **EXPLAIN**
  - **EXPLAIN AVOID_EXECUTE**

• Join Order Directives
  – FROM절에 나열된 테이블 순서대로 join 한다 (Rule based)

```
SELECT --+ ORDERED
       c.customer_num, sum(total_price)
  FROM customer c, orders o, items i
 WHERE c.customer_num = o.customer_num
   AND o.order_num = i.order_num
 GROUP BY 1;
```

• Optimization Goal Directives

```
SELECT --+ FIRST_ROWS
       FIRST 50 *
  FROM customer;
```

• Explain Directives
  – 해당 SQL에 대해서 SET EXPLAIN 구문과 같은 역할을 한다

```
SELECT --+ EXPLAIN
       a.*
  FROM customer a, orders b
 WHERE a.customer_num = b.customer_num ;

SELECT --+ EXPLAIN AVOID_EXECUTE
       a.*
  FROM customer a, orders b
 WHERE a.customer_num = b.customer_num ;
```

The SQL Statement Cache (SSC)

- **SQL문장을 Shared memory에 cache 시킬 수 있다**
  - **사용자 세션의 메모리 절약**
  - **Cache된 SQL구문은 re-parse되지 않는다**

- **서버 단위의 SSC 설정**
  - **$ONCONFIG에서 다음 파라매터 추가 후 서버 재 기동**
    STMT_CACHE 1
  - **onmode 유틸리티로 동적 설정**
    onmode –e enable (세션 단위로 SSC를 설정해야 한다)
    onmode –e on

- **세션 단위의 SSC 설정**
  - **환경변수**
    export STMT_CACHE=1
  - **SQL문**
    SET STATEMENT_CACHE ON;

- SSC(SQL Statement Cache)가 enable되어 있으면 실행하는 모든 SQL문장에 대해 cache된 SQL구문을 비교한다. SQL이 대소문자 및 공백 문자열까지 정확히 일치하는 경우에 cache된 SQL을 사용하게 된다.
- SSC가 적용될 수 있는 SQL문은 다음과 같은 조건을 만족해야 한다
  - DML 문 (SELECT, UPDATE, DELETE, INSERT)
  - Built-in type 및 operator 만 가능
  - UDR(User Defined Routines 및 Procedure)은 불가능
  - 분산 테이블 및 임시 테이블은 불가능
  - select list안에 다른 select문은 사용 불가능
  - union으로 연결된 여러 SQL구문은 사용 불가능
  - Server API를 사용한 경우 cache 불가능
- SSC는 대규모의 사용자 환경에서 동일한 SQL을 반복적으로 처리하는 정형화된 어플리케이션을 수행하는 경우 유용하다

onmode	설 명	비 고
-e on	SSC를 항상 활성화	$ONCONFIG에서 STMT_CACHE 2
-e off	SSC를 사용하지 않는다	$ONCONFIG에서 STMT_CACHE 0
-e enable	SSC를 활성화 하지만 세션에서 SSC를 사용하도록 지정하여야 적용된다	$ONCONFIG에서 STMT_CACHE 1 • 세션 환경 변수 설정 export STMT_CACHE=1 • SQL구문 SET STATEMENT CACHE ON ;
-e flush	사용하지 않는 cache된 SQL을 삭제한다	

The SQL Statement Cache (SSC)

- **Monitoring the SSC**
  - **onstat –g cac stmt**
  - **export STMT_CACHE_DEBUG=<level>:<filename>**
    level = 1 : basic debugging
    level = 2 : extended debugging

• onstat –g cac stmt는 onstat –g ssc와 같다

```
Informix > onstat –g cac stmt
Statement Cache Summary:
#lrus   currsize  maxsize   Poolsize  #hits   nolimit
4       14336     524288    20480     0       1
Statement Cache Entries:
lru hash ref_cnt hits flag heap_ptr      database          user
--------------------------------------------------------------------------------
  0   45       0    2  -F  b164690      stores_demo       informix
  select * from customer

  1   82       1    0   -F  b355420      stores_demo
  select * from CUSTOMER

    Total number of entries: 2.
```

hits 수가 높을 수록 cache된 SQL을 많이 사용하므로 메모리 사용율과 성능이 좋다

동일한 SQL이지만 대소문자를 구분하므로 별도의 entry로 cache된다

• 환경변수 STMT_CACHE_DEBUG를 세션별로 설정하여 cache된 SQL을 사용하는 상태를 파일로 로깅할 수 있다. 지정한 파일명 뒤에 자동으로 세션번호가 붙는다.

```
Informix > export STMT_CACHE_DEBUG=2:/tmp/ssc.out

Informix > ls /tmp/ssc.out*
/tmp/ssc.out435
Informix > vi /tmp/ssc.out435
SSC identity (OK:status=1):  select * from customer
SSC link (OK):  select * from customer
SSC identity (OK:status=1):  select * from customer
SSC link (OK):  select * from customer
SSC identity (stmt not found):  select * from Customer
SSC qualify (OK): select * from Customer
SSC key insert (OK):  select * from Customer
SSC full insert (OK):  select * from Customer
```

```
select * from customer;
select * from customer;
select * from Customer;
select * from Customer;
```

• SQL문장을 cache하는 방법 외에 테이블데이터나 인덱스 자체를 cache할 수 있다. (2장의 Performance Tuning for Tables, Performance Tuning for Indexes 참조)

```
SET TABLE table_name MEMORY_RESIDENT;
SET TABLE table_name NON_RESIDENT;
SET INDEX index_name MEMORY_RESIDENT;
SET INDEX index_name NON_RESIDENT;
```

Update Statistics

- **UPDATE STATISTICS구문을 사용하여 시스템 카타로그 테이블의 통계 정보를 갱신한다**

- **구문**
  - **데이터베이스의 모든 오브젝트에 대하여 수행**
    UPDATE STATISTICS [LOW | MEDIUM | HIGH] ;
  - **특정 테이블과 그 테이블에 생성된 인덱스에 대하여 수행**
    UPDATE STATISTICS [LOW | MEDIUM | HIGH]
    FOR TABLE [table_name] ;
  - **특정 컬럼에 대하여 수행**
    UPDATE STATISTICS [LOW | MEDIUM | HIGH]
    FOR TABLE table_name (column_name) ;

- Optimizer가 query를 실행하기 전에 적은 cost의 query plan을 결정하기 위하여 시스템 카타로그 테이블의 통계 정보를 참조하는데, 이 정보는 UPDATE STATISTICS 구문을 실행하기 전까지는 실제 데이터의 정보를 자동적으로 갱신하지 않으므로 주기적으로 실행시켜야 한다.

- UPDATE STATISTICS 실행 범위
  - 데이터베이스의 모든 오브젝트
  - 특정 테이블과 그 테이블에 생성된 인덱스
  - 모든 테이블과 인덱스 (FOR TABLE 구문 뒤에 table 이름이 없을 경우)
  - 특정 컬럼

- UPDATE STATITSTICS 모드
  - HIGH : 테이블,인덱스,컬럼에 대한 통계 및 정확한 데이터의 분포 (distribution) 빈(bin)을 생성
  - MEDIUM : 테이블, 인덱스, 컬럼에 대한 통계 및 85%~99%정도의 정확도를 가진 샘플링 데이터에 대해 distribution bin을 작성
  - LOW : distribution bin없이 테이블, 인덱스, 컬럼에 대한 통계 작성 (default)

- 시스템 카타로그 테이블의 통계 정보 내용
  - systables
    - nrows : 테이블의 데이터 row수
    - npused : 테이블의 사용된 디스크 페이지 수
  - sysindexes
    - leaves : 인덱스 B+트리의 leaf 페이지 수
    - levels : 인덱스 B+트리의 level 수
    - nunique : unique한 key값의 수 (composite인덱스의 경우 첫번째 컬럼값만 반영)
    - clust : 데이터 페이지의 클러스터링 정도 (값이 작을수록 클러스터링이 좋다)
  - syscolumns
    - colmin : 컬럼의 두번째로 적은 값
    - colmax : 컬럼의 두번째로 큰 값

Distributions

- **UPDATE STATISTICS의 MEDIUM이나 HIGH 모드는 데이터 분포에 대한 정보를 시스템 카타로그에 기록한다**

- **Distribution 정보 생성 단계**
  - **1. 컬럼 데이터 조회**
  - **2. 컬럼 데이터 정렬**
  - **3. 컬럼 데이터 분포 정보 생성**

- **Distribution 정보**
  - **시스템 카타로그 테이블 sysdistrib 에 저장**
  - **dbschema 유틸리티를 통하여 정보 조회**
    dbschema –d db_name –hd { all | table_name }

- 데이터 분포(distribution) 정보는 시스템 카타로그 테이블 sysdistrib에 저장된다.
- 데이터 샘플링 방법을 사용하는 MEDIUM모드에 비해 HIGH 모드는 시스템 리소스와 수행시간을 좀 더 많이 사용하게 되므로 사이즈가 큰 테이블에 대해서는 MEDIUM 모드가 적당하지만 정확도는 다소 떨어질 수 있다.
- LOB 자료형에 대해서는 데이터분포 정보를 생성하지 않는다.

- Distribution 정보 생성 단계
  - 컬럼 데이터 조회
    - 특정 컬럼에 대한 UPDATE STATISTICS 구문을 실행하고 그 컬럼이 인덱스의 첫번째 컬럼 이라면 인덱스 페이지만 조회
    - 그렇지 않은 경우 모든 데이터 페이지를 조회
  - 컬럼 데이터를 정렬(sort) 루틴에 전송
    - MEDIUM모드일 경우 컬럼 데이터의 일부만 전송
    - 이러한 샘플 데이터의 양은 resolution과 confidence에 의해 결정
  - 데이터 정렬
    - 만약 첫번째 단계에서 인덱스 페이지를 조회했다면 이미 데이터가 정렬되어 있으므로 이 단계는 통과
  - 데이터 분산 정보 수집 저장
    - 데이터의 매 N번째 마다 첫번째 값, 마지막 값 수집 (N = resolution/100 * 데이터 행)
    - 이러한 각각의 정보는 각각 bin이라 불리는 단위로 구분하여 저장
    - 중복값이 많은 데이터에 대한 정보는 overflow bin으로 저장

```
Informix > dbschema –d stores_demo –hd all

--- DISTRIBUTION ---
    (                          0)
  1: (  5638,   16,           16)
  2: (  5638,    9,           26)
  3: (  5638,    4,           31)
  4: (  5638,    4,           36)
  5: (  5638,    4,           41)
  6: (  5638,    4,           46)
  7: (  5638,    4,           51)
  8: (  5638,    4,           56)
  9: (  5638,   13,           69)
 10: (  3145,   29,           99)

--- OVERFLOW ---
  1: (  6112,                 80)
```

한 bin 에 기록된 데이터 수

한 bin 안에서 unique한 값의 수

한 bin 안에서의 최대값

중복값의 수

많은 중복값

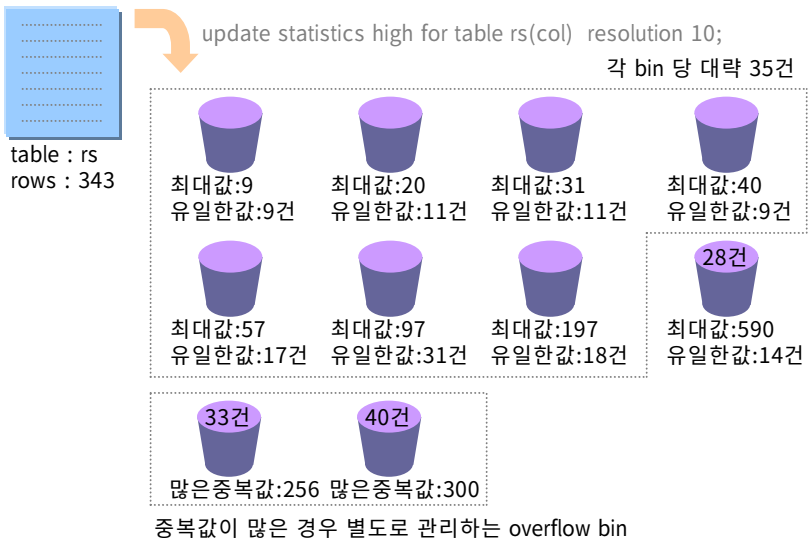
Resolution / Confidence

- **Resolution**
  - **데이터 distribution 정보를 작성할 때 몇 개의 bin으로 나눌 지 결정**
  - **100 / resolution = # bins**

```
UPDATE STATISTICS HIGH FOR TABLE table_name
        RESOLUTION 10 ;
```

- **Confidence**
  - **MEDIUM 모드일 때 sample 데이터에 대한 신뢰도(reliability)**

```
UPDATE STATISTICS MEDIUM FOR TABLE table_name
        RESOLUTION 10 .99 ;
```



- Resolution은 0.005 ~ 10 사이의 수인데, 1/전체row수 보다 작을 수는 없다
- HIGH모드의 resolution 기본값은 0.5, MEDIUM의 resolution 기본값은 2.5
- Resolution이 작으면 bin이 많아지고 각 bin에는 적은 수의 데이터에 대한 정보를 포함하므로 optimizer가 좀 더 정확하게 query path를 결정할 수 있으나 상대적으로 많은 distribution 정보를 읽어야 하기 때문에 optimizing 시간이 길어 질 수 있다
- 예)
  - Resolution이 1이면 100개의 bin을 만들고 각 bin에 전체 데이터의 1%의 정보를 기록한다.
  - Resolution이 10이면 10개의 bin을 만들고 각 bin에 전제 데이터의 10%의 정보를 기록한다.

- Confidence는 .80 ~ .99 사이의 값으로 .99는 99%의 신뢰도를 의미한다.
- Confidence는 MEDIUM모드에서만 사용되고 기본값은 .95이다

```
UPDATE STATISTICS MEDIUM
    FOR TABLE rs (col1)
    RESOLUTION 10 .80 ;
```

```
Informix > dbschema –d stores_demo
                    –hd rs
Medium Mode, 10.000000 Resolution,
0.800000 Confidence
--- DISTRIBUTION ---
    (                       1)
 1: (  34,    18,           8)
 2: (  34,    27,          20)
 3: (  34,    24,          30)
 4: (  34,    27,          41)
 5: (  34,    33,          61)
 6: (  34,    33,         152)
 7: (  34,    30,         199)
 8: (  30,    24,         590)
--- OVERFLOW ---
 1: (  12,                 6)
 2: (  24,                256)
 3: (  36,                300)
```

```
UPDATE STATISTICS MEDIUM
    FOR TABLE rs (col1)
    RESOLUTION 10 .99 ;
```

```
Informix > dbschema –d stores_demo
                    –hd rs
Medium Mode, 10.000000 Resolution,
0.990000 Confidence
--- DISTRIBUTION ---
    (                       1)
 1: (  34,    12,           9)
 2: (  34,    14,          20)
 3: (  34,    14,          31)
 4: (  34,     9,          39)
 5: (  34,    19,          53)
 6: (  34,    30,          91)
 7: (  34,    26,         197)
 8: (  30,    18,         590)
--- OVERFLOW ---
 1: (  36,                256)
 2: (  41,                300)
```

Update Statistics Guideline

- **데이터베이스 전체 또는 테이블별로 LOW 모드로 수행하는 것이 일반적**
- **LOW 모드로 수행하였을 때 만족스럽지 못하다면 다음 단계 수행**
  - **대용량 테이블의 경우**
    1. 테이블별로 MEDIUM 모드, distribution only 옵션으로 수행
    2. 인덱스의 첫번째 컬럼에 대하여 HIGH 모드 수행
    3. Multi-column으로 생성된 인덱스 고려
    4. 기타 컬럼에 대하여 컬럼 단위로 LOW 모드 수행
  - **소규모 테이블의 경우**
    테이블 단위로 HIGH 모드 수행

DISTRIBUTIONS ONLY 옵션을 추가하면 systables, sysindexes, syscolumns의 정보를 수집하지 않고 단지 데이터 분포도에 대한 정보만 갱신하게 된다.

데이터 분포 정보를 삭제하려면 UPDAE STATISTICS LOW 수행을 하면서 DROP DISTRIBUTIONS 옵션을 추가한다

UPDATE STATISTICS LOW DROP DISTRIBUTIONS ;
UPDATE STATISTICS LOW FOR TABLE table_name DROP DISTRIBUTIONS ;

DROP DISTRIBUTIONS 옵션 없이 UPDATE STATISTICS LOW만 수행하면 기존에 생성된 데이터 분포에 대한 정보는 그대로 둔 채 LOW모드의 통계 정보 (즉, systables, sysindexes 등)만 갱신하게 된다.

UPDATE STATISTICS LOW FOR TABLE table_name (column)
DROP DISTRIBUTIONS ;
위와 같이 사용하면 지정한 컬럼에 대하여 데이터 분포 정보는 삭제되고 새로 systables, sysindexes, syscolumns와 관련된 통계 정보를 갱신한다

컬럼의 타입이나 크기가 변경되거나 컬럼이 삭제되면 그 컬럼과 관련된 데이터 분포 정보도 삭제 된다.

- 일반적으로 데이터베이스 전체에 LOW 모드 (default)로 통계 정보 갱신
  UPDATE STATISTICS ; 또는
  UPDATE STATISTICS LOW ;
- 데이터베이스가 커서 한번에 수행되기 힘들 경우 테이블 단위로 나누어 실행
  UPDATE STATISTICS FOR TABLE table_name ;

- 보통 LOW모드로 통계 정보를 갱신하여도 query 수행에 상당한 효과를 볼 수 있으나 좀 더 정확한 optimizing 정보를 제공하기 위하여 추가적으로 실행할 수 있다. 테이블 크기가 작은 경우라면 다음과 같이 HIGH모드를 사용한다 (resolution은 1.0, confidence는 .99 정도 적당)
  UPDATE STATISTICS HIGH FOR TABLE table_name ;
- 하지만 HIGH 모드는 LOW모드에 비해 UPDATE STATISTICS 수행 시간이 길어지므로 대용량의 테이블에 대해서는 다음과 같은 단계로 수행한다
  – 모든 테이블에 대하여
  UPDATE STATISTICS MEDIUM FOR TABLE table_name DISTRIBUTIONS ONLY ;
  – 인덱스의 첫번째 컬럼에 대하여
  UPDATE STATISTICS HIGH FOR TABLE table_name(column) ;
  – 동일한 컬럼으로 시작되는 composite index가 여러 개 있을 때, 처음으로 나오는 다른 컬럼에 대하여, 즉 다음과 같이 인덱스가 생성되어 있을 때, c 컬럼과 e컬럼에 대하여
  CREATE INDEX ix1 ON table1 (a,b,c,d);
  CREATE INDEX ix2 ON table1 (a,b,e,f);
  UPDATE STATISTICS HIGH FOR TABLE table1 (c) ;
  UPDATE STATISTICS HIGH FOR TABLE table1 (e) ;
  – 기타 다른 모든 컬럼에 대하여 LOW모드로 수행
  UPDATE STATISTICS LOW FOR TABLE table_name(column) ;

Update Statistics Guideline

- **다음과 같은 경우 반드시 UPDATE STATISTICS를 수행해야 한다**
  - **데이터를 테이블에 대량 로드 한 후**
  - **데이터 수정이 상당히 많아서 데이터 분포가 변경 되었을 때**
  - **테이블의 대부분의 행에서 입력 또는 삭제 작업이 일어났을 때**
  - **인덱스를 추가하거나 삭제 또는 변경하였을 때**

```
-- 많은 테이블에 대하여 쉽게 UPDATE STATISTICS 수행 구문을 만들 수 있다
OUTPUT TO "updstats.sql" WITHOUT HEADINGS
SELECT UNIQUE "update statistics high for table " ||
       TRIM (tabname) || "(" || trim(colname) || ")" ,
       " distribution only ; "
  FROM systables , sysindexes, syscolumns
 WHERE systables.tabid = sysindexes.tabid
   AND systables.tabid = syscolumns.tabid
   AND sysindexes.part1 = syscolumns.colno;
```

- Query tuning의 방법으로 SQL구문에 대한 tuning외에 UPDATE STATISTICS를 사용한 데이터베이스 tuning 방법이 있다
  - 1. SQL구문에 SET EXPLAIN ON을 추가하여 query plan을 기록한다
  - 2. where절에 포함된 컬럼에 대하여 UPDATE STATISTICS HIGH 수행
  - 3. 1번 과정을 반복
  - 4. 1번과 3번 출력 결과에서 cost와 query plan을 비교한다. cost가 적게 나온다면 optimizer가 좀 더 나은 query plan을 결정하였다는 것을 알 수 있다.
  - 5. HIGH모드는 UPDATE STATISTICS수행 시간 자체가 길어지므로 가능하면 MEDIUM 모드로 수행한 상태에서도 동일한 query plan을 작성하도록 하는 것이 바람직하다. 따라서 confidence 를 .99, resolution을 증가 시켜가며 UPDATE STATISTICS MEDIUM 을 수행한다
  - 6. 1번 과정을 반복한다
  - 7. 3번 과정, 즉 HIGH모드로 수행했을 때와 동일한 query plan을 가지는지 확인하고 그렇지 않다면 동일한 query plan이 되는 resolution을 찾을 때까지 반복한다.

- 참고) query의 수행시간은 OS 프롬프트 상에서 timex와 같은 unix 명령어를 사용할 수도 있지만, query를 수행하는 session에서 환경변수 SQLSTATS=2 를 먼저 설정하면 sysmaster의 syssqlcurses 테이블에서 현재 active상태의 세션의 query 수행시간을 조회할 수 있다. (일부 버전은 지원되지 않음)
  select scs_sessionid, scs_currdb, scs_cumtime, scs_sqlstatement
  from syssqlcurses;
- 참고) DBUPSPACE환경변수는 UPDATE STATISTICS를 수행하는 동안 데이터 정렬을 위한 임시 디스크 공간의 크기를 지정한다. 이 환경변수를 지정하지 않으면 서버가 필요한 크기를 판단하여 할당하는데, 다른 사용자 작업 등을 위해 임시 영역 크기를 제한하려는 경우 사용한다. 5000KB보다 적은 값으로 환경변수에 할당하는 경우 최소 5000KB만큼은 사용된다.
  export DBUPSPACE=10000 (10M만큼 할당)
- 참고) Procedure에 대하여 통계 정보를 갱신하려면 다음과 같이 실행한다
  UPDATE STATISTICS FOR PROCEDURE ;
- 참고) PDQ를 설정하고 통계 정보를 갱신하면 좀 더 빠르게 수행할 수 있지만 PDQ를 설정한 상태에서 procedure에 대한 통계 정보를 갱신하면 procedure가 실행될 때 먼저 PDQ를 설정하고 procedure logic을 수행하게 되므로 OLTP환경에서는 적합하지 않다.

5. Data Migration

Data Migration Tools
SQL Load / SQL Unload
dbload Utility
dbexport utility
dbimport utility
onload / onunload utility
HPL (High Performance Loader)

Data Migration Tools

- **SQL Load / SQL Unload**
- **dbload**
- **dbexport / dbimport**
- **onload / onunload**
- **HPL(High Performance Loader)**

• 인포믹스에서 제공하는 migration tool은 다음과 같은 특징이 있다

	load , unload	dbload	dbexport , dbimport	onload , onunload	HPL
속도	★☆☆	★☆☆	★☆☆	★★☆	★★★
파일 형식	ASCII	ASCII	ASCII	Binary	ASCII / Binary
Delimiter	지원	지원	지원	N/A	지원
Fixed Length		지원		N/A	지원
단위	테이블	테이블	데이터 베이스	데이터 베이스, 테이블	테이블
비고		unload 기능 없음. 다양한옵션			Deluxe / Express 모드

SQL Load / SQL Unload

- **SQL의 확장 구문**
- **Delimiter가 있는 파일로 unload / load**
- **Delimiter의 기본값은 "|"**
  - **unload / load 문장의 DELIMITER 옵션이나**
  - **DBDELIMITER 환경변수로 기본값 변경 가능**
- **구문**

UNLOAD TO 'file_name' [DELIMITER 'delimiter_char']
SELECT 구문 ;
LOAD FROM 'file_name' [DELIMITER 'delimiter_char']
INSERT INTO table_name [(column_list)] ;

- DBDELIMITER환경변수가 없는 경우 DELIMITER의 기본값은 파이프 기호('|')
- 다음과 같은 DELIMITER는 사용할 수 없다
  - 역 슬래시 (\ )
  - newline 문자
  - hex 값
- DELIMITER는 공백,탭 등 모든 문자가 가능하다
  export DBDELIMITER=' '
  export DBDELIMITER='	'
- Unload구문을 사용할 때 다양한 SQL 구문을 사용할 수 있으므로 특정 컬럼 또는 특정 조건에 맞는 레코드만 unload 할 수 있다

```
CREATE TABLE unl_table (col1 INT, col2 CHAR(20), col3 DATE);
INSERT INTO unl_table VALUES (100,'first',TODAY);
INSERT INTO unl_table VALUES (200,'second',TODAY);
INSERT INTO unl_table VALUES (300,'third',TODAY);

UNLOAD TO unload_file DELIMITER ',' SELECT * FROM unl_table; -- 3건 unload
LOAD FROM unload_file DELIMITER ',' INSERT INTO unl_table; -- 3건 load
-- 총 6건
UNLOAD TO unload_file SELECT col1, col2 FROM unl_table; -- 6건 unload
LOAD FROM unload_file INSERT INTO unl_table (col1, col2); -- 6건 load
-- 총 12건
```

- Load 팁
  - Load구문은 기존 테이블에 데이터를 계속 append한다
  - Serial컬럼값을 자동으로 증가시키려면 데이터 파일에 0를 넣는다
  - DELIMITER사이에 아무런 값도 없으면 NULL이 입력된다
  - Logging모드 DB에서 load문은 하나의 트랜잭션이므로 long transaction이 발생될 수 있다. 따라서 먼저 no-logging 모드로 변경하고 load하거나 데이터를 나누어서 load하는 것이 좋다
  - No-logging모드 DB에서 load문이 중간에 fail이 나면 그 시점까지의 데이터가 load된 상태로 남는다 (Logging모드에서는 모두 rollback된다)
- Unload 팁
  - Unload 구문은 기존에 동일한 이름의 파일이 존재하는 경우 기존의 데이터 파일을 overwrite한다
  - VARCHAR컬럼 데이터의 후행공백은 유지된다
  - Delimiter와 같은 문자 데이터가 unload되면 delimiter와 구분하기 위하여 그 문자 앞에 역슬래시(\ )가 붙는다

SQL Load / SQL Unload

- **Simple Large Object**
  - **TEXT나 BYTE컬럼 데이터가 unload 파일에 함께 unload된다**
  - **Byte 값은 ASCII-Hexadecimal 형식으로 저장된다**
  - **Byte는 선행 또는 후행 공백 없이 load된다**
- **Smart Large Object**
  - **CLOB이나 BLOB 데이터가 별도의 파일로 저장된다**
  - **Unload파일에는 별도로 저장된 CLOB, BLOB데이터에 대한 정보를 가지고 있다**
    - **large object 안에서 데이터의 시작 위치**
    - **large object 의 크기**
    - **large object 를 저장한 파일 이름**
- **SQL unload / load 구문은 단순하므로 간단한 Shell 프로그램을 작성하기에 용이하다**

• Simple Large Object 컬럼을 포함한 unload 파일의 예

```
Informix> dbaccess stores_demo << !
UNLOAD TO simple_file SELECT * FROM simple;
!
Informix > vi simple_file
100|여기는 \
텍스트 \
컬럼 자리|2002/12/12|
200|파이프 기호 \| 입력|2002/12/12|
```

• Smart Large Object 컬럼을 포함한 unload 파일의 예

```
Informix> dbaccess stores_demo << !
UNLOAD TO smart_file SELECT * FROM smart;
!
Informix > vi smart_file
100|0,18,clob9ce7.394|2002/12/12|
200|18,18,clob9ce7.394|2002/12/12|
Informix > vi clob9ce7.394
여기는 CLOB
자료형 입력
여기는 CLOB
자료형 입력
```

• SQL unload를 이용한 간단한 데이터베이스 테이블 백업 쉘 프로그램

```
#!/bin/sh

dbaccess stores_demo << !
        UNLOAD TO table_name.list DELIMITER ' '
        SELECT tabname
          FROM systables
         WHERE tabid > 99
           AND tabtype = 'T';
!

for table in `cat table_name.list`
do
        dbaccess stores_demo << !
                UNLOAD TO $table.txt DELIMITER ','
                 SELECT * FROM $table ;
!
done

rm -f table_name.list
```

dbload Utility

- **SQL load 문보다 확장된 기능을 제공하는 load 유틸리티**
- **Delimiter형과 fixed length형 파일 모두 지원**
- **command 파일을 작성하여야 한다**
- **dbload –d database_name**
  –c command_file
  –l log_file
  [–e error_number]
  [–n commit_row]
  [–i ignore_number]
  [–r | –k]
  - **필드 매핑 기능**
  - **NULL 값 표시**
  - **최대 오류 건수 지정**
  - **commit 간격 건수**
  - **데이터 파일의 시작 위치 지정**
  - **로드 중 lock 레벨 선택**

• SQL load와 동일한 형태의 dbload 예

```
Informix> cat command_file.txt
FILE sample.txt DELIMITER '|' 4 ;
INSERT INTO sample ;
Informix> dbload –d stores_demo –c command_file.txt –l error_history.log
```

하나의 command 파일에 여러개의 FILE~INSERT 구문을 넣을 수 있다

데이터 파일

delimiter의 종류와 컬럼 개수 지정

error가 발생하면 error내용을 기록

• dbload 팁
  – 너무 복잡한 command file을 사용하면 속도가 떨어질 수 있다
  – 중소 규모의 데이터 로드에 적합하다
  – 연속되지 않은 데이터 필드를 하나의 컬럼에 로드 할 수 있다
  – 하나의 데이터 파일을 여러 테이블에 로드 할 수 있다

• dbload 옵션
  – Load 중에 10건의 row가 에러가 나면 dbload가 멈추게 되는데 (기본값), dbload를 종료하기 전까지 허용하는 에러 row의 수를 –e 옵션으로 지정할 수 있다
  dbload –d database_name –c command_file –e 500
  (500건의 에러가 발생하면 dbload 중단)
  – SQL load문과는 달리, logging 모드 데이터베이스에서 10건이 로드되면 commit하여(기본값) long transaction을 방지하는데, -n 옵션을 사용하여 적당한 commit간격을 결정할 수 있다
  dbload –d database_name –c command_file –n 20000
  (20000건 로드될 때마다 commit)
  – dbload를 수행하다 중단되면 데이터의 처음 위치부터 다시 로드할 필요 없이 이미 로드된 행은 건너 뛰어 새로 로드 할 수 있다. 예를 들어 데이터 파일의 282번째 행까지 로드 된 후 dbload가 중단되면 283번째 행부터 다시 로드를 시작해야 하는데, 다음과 같이 수행한다
  dbload –d database_name –c command_file –i 282
  – dbload가 수행되는 동안 테이블의 row단위로 lock이 걸리는데 (-r옵션) row 레벨 lock은 동시성은 우수하지만 lock에 대한 자원 할당이 빈번하게 된다. –k 옵션을 사용하게 되면 테이블 단위로 exclusive lock을 사용하는데, 동시성은 떨어지지만 대량의 파일을 로드할 때 lock 자원이 절약된다.
  dbload –d database_name –c command_file -k

dbload Utility

- **Example**
  - **Delimiter Form**
  - **Character-Position Form**

• schema

```
CREATE TABLE delimiter_form (
        id      INT,
        flag    CHAR(14),
        amount  DECIMAL(10,2),
        memo    CHAR(20)
);
```

```
CREATE TABLE char_position1 (
        number  INT,
        phone   CHAR(12),
        code1   CHAR(3),
        code2   CHAR(4)
);
CREATE TABLE char_position2 (
        number  INT,
        code3   CHAR(7)
);
```

• Delimiter form

```
Informix> cat delimiter_form.cmd
FILE delimiter_form.dat DELIMITER ',' 3;
INSERT INTO delimiter_form (id, flag, amount, memo)
VALUES (f03, 'constant_value', f02, f01);

Informix> dbload -d stores_demo -c delimiter_form.cmd -l error.log
```

delimiter종류와 개수 지정

데이터파일의 데이터는 한 레코드에서 각 필드가 f01,f02,f03등으로 참조된다

delimiter_form.dat

```
TXB200,1000.20,1,
AWZ000,5400.10,2,
```

dbload

stores_demo

delimiter_form

1	constant_value	1000.20	TXB200
2	constant_value	5400.10	AWZ000

• Character-Position Form

```
Informix> cat char_position.cmd
FILE char_position.dat
(       number  20-22,
        phone   4-15    NULL='xxx-xxx-xxxx',
        code1   1-3,
        code2   16-19   NULL='xxxx',
        code3   16-19 : 4-6
);
INSERT INTO char_position1 VALUES (number,phone,code1,code2);
INSERT INTO char_position2 VALUES (number,code3);

Informix> dbload -d stores_demo -c char_position.cmd -l error.log
```

데이터파일에서 데이터의 위치로 레코드 필드를 구분

지정한 문자열이 데이터 파일에 있으면 null값으로 처리

필드의 위치 조합 가능

여러개의 insert문 가능

char_position.dat

```
WRG076-252-6235AEYT001
RUH066-910-3968xxxx002
TQLxxx-xxx-xxxxJKOE003
```

dbload

stores_

char_position1

1	076-252-6235	WRG	AEYT
2	066-910-3968	RUH	null
3	null	TQL	JKOE

char_position1

1	AEYT076
2	xxxx066
3	JKOExxx

dbload Utility

- **Example**
  - **row types 컬럼이 포함된 table**
  - **named row type으로 생성된 typed table**
  - **collections type 컬럼이 포함된 table**

• schema

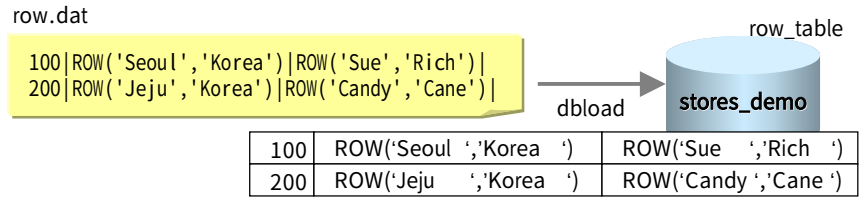
```
CREATE ROW TYPE named_row (
    address1 char(20) NOT NULL,
    address2 char(20)
) ;

CREATE TABLE row_table (
    number  integer,
    address named_row,
    name    ROW (
            fname CHAR(20),
            lname CHAR(20) NOT NULL
    )
);
```

```
CREATE ROW TYPE cust_t (
    fname   VARCHAR(30),
    lname   VARCHAR(30) NOT NULL,
    state   VARCHAR(2),
    zip     VARCHAR(6)
);

CREATE TABLE typed_table OF TYPE cust_t;
```

```
CREATE TABLE collection (
    name    CHAR(30),
    children SET(VARCHAR(30) NOT NULL)
);
```

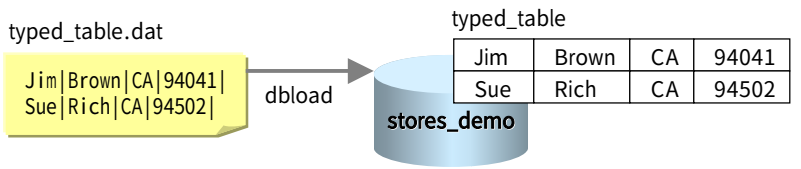
• row types

```
Informix> cat row.cmd
FILE row.dat DELIMITER '|' 3;
INSERT INTO row_table;

Informix> dbload -d stores_demo -c row.cmd -l error.log
```



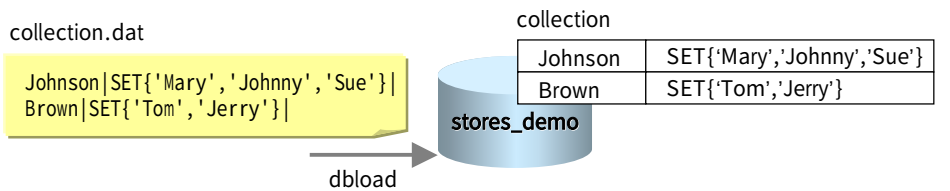
• typed table

```
Informix> cat typed_table.cmd
FILE typed_table.dat DELIMITER '|' 4;
INSERT INTO typed_table;

Informix> dbload -d stores_demo -c typed_table.cmd -l error.log
```



• collections

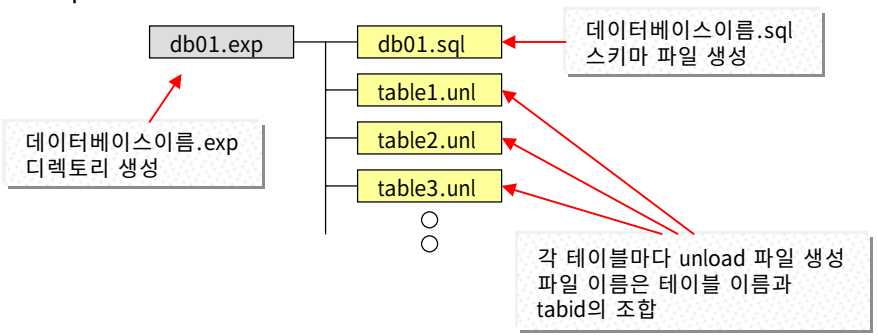
```
Informix> cat collection.cmd
FILE collection.dat DELIMITER '|' 2;
INSERT INTO collection;

Informix> dbload -d stores_demo -c collection.cmd -l error.log
```



dbexport utility

dbexport
- **전체 데이터베이스를 ASCII형태의 파일로unload**
- **전체 데이터베이스의 schema를 파일로 작성**
- **dbexport <데이터베이스> [–q] [–ss]**
  [
  { –o <directory> |
  –t <tape device> –b <block size> –s <tape size> [–f <sql file>] }
  ]

• dbexport 파일 구조
dbexport db01 –ss



• dbexport의 특징
  – dbexport를 수행하는 동안 데이터베이스에 exclusive lock 발생
    • 다른 사용자가 데이터베이스에 connect하여 사용하고 있는 동안 dbexport 수행 불가능
    • dbexport를 수행하는 동안 다른 사용자가 데이터베이스에 connect 불가능
  – 각 테이블을 순차적으로 unload
  – 데이터베이스에 DBA권한이 있는 사용자가 실행 가능

• dbexport 옵션
  – q : 진행 과정이 화면에 출력되지 않는 quiet mode. dbexport.out 파일에만 기록됨
  – ss : Server specific 옵션. 다음과 같은 정보가 출력된다
    • 테이블 또는 인덱스의 dbspace 옵션
    • first , next extent 크기
    • 테이블의 lock 모드
  – o : DB명.exp가 생성될 디렉토리 지정 (default는 현재 디렉토리)
  – t ,b, s : 테이프 장치를 사용할 경우
  – f : 테이프 장치 옵션과 함께 사용, 데이터는 테이프로 보관하고 스키마 파일(DB명.sql)은 별도의 파일로 저장

• text,byte가 포함된 데이터베이스를 dbexport, dbimport할 때 환경변수 DBTEMP로 지정된 파일 시스템 영역 또는 /tmp 영역이 충분하여야 한다

• dbexport 사용 예

```
# 현재 디렉토리에 stores_demo export
Informix> dbexport stores_demo –ss

# 지정한 테이프 장치에 stores_demo export, 스키마는 별도의 파일로 생성
Informix> dbexport stores_demo –t /dev/rmt0 –b 1024 –s 102400 –f /tmp/db.sql

# 지정한 디렉토리에 stores_demo export
Informix> dbexport stores_demo –o /data -ss
```

dbimport utility

- **dbimport**
  - **dbexport로 unload 받은 데이터를 이용하여 데이터베이스 및 테이블 등 전체 schema를 생성하고 데이터를 load**
  - **dbimport <데이터베이스> [–q]**
    [–d <dbspace>] [–l [buffered]]
    [
    { –i <directory> |
    –t <tape device> –b <block size> –s <tape size> [–f <sql file>] }
    ]

- **dbexport/ dbimport 는 데이터베이스 단위로 서로 다른 시스템, IDS 버전간 데이터 migration 에 유용하게 사용된다**

• dbimport의 특징
  – dbimport 수행 중에는 데이터베이스에 exclusive lock 발생
  – dbimport는 데이터베이스를 새로 생성하므로 해당 데이터베이스가 존재하지 않을 때 수행가능
  – 테이블을 순차적으로 load
  – 데이터 load 후 index 생성

• dbimport 옵션
  – d : 데이터베이스가 생성될 dbspace 지정
  – l [buffered] : 데이터베이스의 로깅 모드 설정 (unbuffered, buffered)
  – i : 현재 디렉토리가 아닌 다른 디렉토리에 DB명.exp가 있을 경우 지정
  – t, b, s : 테이프 장치 옵션
  – f : dbexport를 수행할 때 -f옵션을 사용하였다면 dbimport시에도 사용

• dbimport guideline
  – PDQPRIORITY를 설정하면 index를 생성할 때 병렬 처리가 된다
  – dbimport할 때 로깅 모드를 설정하지 않고 dbimport가 완료된 후 로깅 모드 전환하는 것이 속도가 향상된다
  – DB명.sql 파일을 수정하여 테이블 및 인덱스의 extent 크기, dbspace 등을 변경할 수 있다
  – 데이터베이스 이름을 변경하고 싶으면 DB명.exp 디렉토리와 DB명.sql을 변경한다.

```
mv old_db.exp new_db.exp
mv new_db.exp/old_db.sql new_db.exp/new_db.sql
```

• dbimport 사용 예

```
Informix> dbimport stores_demo
Informix> dbimport stores_demo –t /dev/rmt0 –b 1024 –s 102400 –f /tmp/db.sql
Informix> dbimport stores_demo –o /data
```

onload / onunload utility

- **onload / onunload**
  - **작업 단위 : 데이터베이스 또는 테이블**
  - **특징 : 데이터 페이지의 바이너리 이미지를 load 또는 unload 하므로 dbexport/dbimport 보다 속도가 빠르다**

  - **onunload [–l]**
    [–t <tape device>] [–b <block size>] [–s <tape size>]
    <database>[:<table>]

  - **onload [–l] [–d <dbspace>]**
    [–t <tape device>] [–b <block size>] [–s <tape size>]
    <database>[:<table>]
    [{ –i <old-index> <new-name> }]
    [{ –fd old-dbspace new-dbspace }]
    [{ –fi index old-dbspace new-dbspace }]
    [{-c <old-constraint> <new-constraint> }]

- 주의사항 : onunload, onload를 사용하여 서로 다른 시스템에 데이터를 이동하려면 두 시스템은 다음과 같은 사항이 만족되어야 한다
  - IDS의 페이지 크기가 동일하여야 한다
  - IDS의 버전이 동일하여야 한다
  - 시스템의 numeric 데이터 포맷이 동일하여야 한다
  - 시스템의 byte alignment가 동일하여야 한다

- 옵션
  - t, b, s : 테이프 장치 옵션
    실제 테이프 장치 대신 파일명으로 테이프 장치를 대신할 수 있다
  - l : 테이프 장치 옵션 대신 $ONCONFIG의 LTAPEDEV, LTAPEBLK, LTAPESIZE 값으로 사용된다 (default 옵션)
  - d : onload로 데이터베이스를 생성할 때 데이터베이스의 생성 위치
  - i, fd,fi, c : 인덱스 이름, dbspace 이름 등을 onload 수행시 변경 가능

- onunload, onload시 lock
  - onunload 수행 동안 각 테이블에 shared lock 발생 (데이터 변경 불가능)
  - onload 수행 동안 각 테이블에 exclusive lock 발생 (조회, 변경 불가능)

onload / onunload utility

- **onunload**
  - **unloading a database**
    - **모든 테이블의 데이터와 인덱스뿐만 아니라 synonym, constraints, default value, trigger, stored procedure 등 보관**
    - **informix 또는 DBA권한의 사용자가 실행**
  - **unloading a table**
    - **해당 테이블의 view, synonym, constraints, default value, trigger는 보관 되지 않는다**
    - **informix 또는 DBA권한의 사용자, 테이블의 owner가 실행**
- **onload**
  - **loading a database**
    - **no logging 모드로 데이터베이스 생성**
    - **onload를 수행하는 사용자가 데이터베이스의 owner가 된다**
  - **loading a table (데이터베이스가 no logging 상태에서 사용)**
    - **테이블 생성**
    - **테이블 또는 인덱스의 dbspace, 인덱스 이름, constraint이름을 변경할 수 있다**

• onunload 예 (데이터베이스)

```
# 지정한 테이프 장치에 stores_demo onunload
Informix> onunload stores_demo –t /dev/rmt0 –b 1024 –s 102400

# 지정한 파일에 stores_demo onunload
Informix> onunload stores_demo –t /backup/unload_file

# $ONCONFIG의 LTAPEDEV로 지정된 장치에 onunload
Informix> onunload stores_demo -l
```

• onload 예 (데이터베이스)

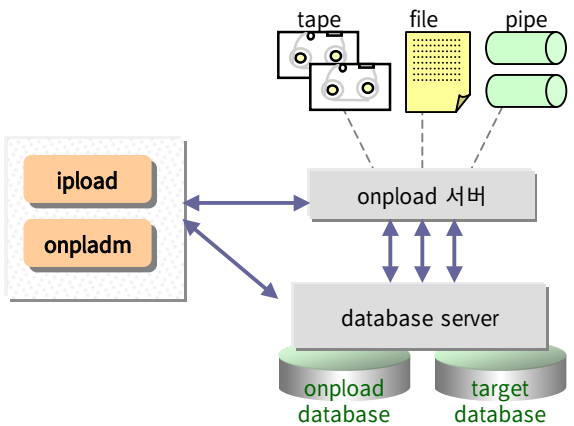
```
# 지정한 테이프 장치에서 stores_demo onload
Informix> onload stores_demo –t /dev/rmt0 –b 1024 –s 102400 –d datadbs01

# 지정한 파일에서 stores_demo onload
Informix> onuoad stores_demo –t /backup/unload_file –d datadbs01

# $ONCONFIG의 LTAPEDEV로 지정된 장치에서 onload
Informix> onload stores_demo –l –d datadbs01
```

• 테이블 onunload , onload 예

```
Informix> dbschema –d stores_demo –t migration –ss
create table migration (
    col1 integer,
    col2 char(40),
    primary key (col1)  constraint const1
  ) in data01 ;
create index migration_index on migration (col2) in data02 ;
Informix> onunload stores_demo:migration –l
Informix> dbaccess stores_demo << !
drop table migration;
!
Informix> onload stores_demo:migration -l -i migration_index m_index  \
          -fd data01 data02  \
          -fi migration_index data02 data01  \
          -c const1 const_new
Informix> dbschema –d stores_demo –t migration –ss
create table migration (
    col1 integer,
    col2 char(40),
    primary key (col1)  constraint const_new
  ) in data02 ;
create index m_index on migration (col2) in data01 ;
```

HPL (High Performance Loader)

- **고성능 데이터 unload, load 툴**
- **구성 요소**
  - **onpload 데이터베이스**
  - **onpload 서버**
  - **ipload 인터페이스(GUI)**
  - **onpladm 인터페이스 (Command)**



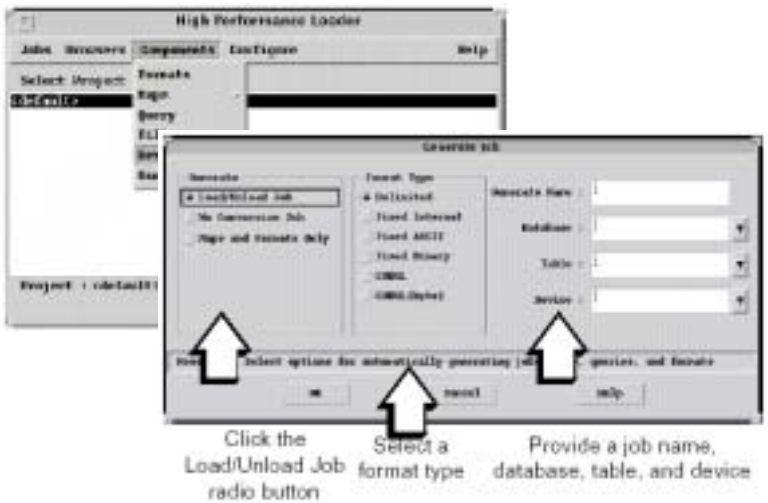
• ipload나 onpladm은 데이터를 로드 또는 언로드하기 위한 작업을 생성하고 실행하는 인터페이스이다.
  – ipload는 UNIX의 Motif 윈도우 환경 또는 Windows의 Motif emulator가 필요하고 DISPLAY환경변수를 설정한다
    export DISPLAY=<ip_address>:0.0
  – onpladm은 9.x 버전 이상에서 제공되는 command line 인터페이스
  – HPL작업을 정의할 때 다음과 같은 오브젝트는 재사용 가능하다
    • Devices : tape, file, pipe
    • Formats : device의 데이터구조
    • Queries(unload job only) : unload될 데이터에 대한 query
    • Maps : 데이터 구조와 실제 테이블 구조와의 매핑
    • Filters(load job only) : load 중 accept되거나 reject될 조건

• onpload 데이터베이스
  – HPL을 최초로 사용하는 시점에 자동적으로 생성.
  – HPL작업을 생성하면 onpload 데이터베이스에 저장.
  – 다른 이름의 HPL 데이터베이스 이름을 사용하고 싶으면 DBONPLOAD 환경변수를 지정한다.
    export DBONPLOAD=new_onpload
  – onpload 데이터베이스는 반드시 target 데이터베이스와 동일한 서버에 생성되어야 한다.

• onpload 서버
  – ipload나 onpladm으로 HPL작업을 실행할 때 실제로 동작되는 process
  – onpload 데이터베이스에서 요청한 작업에 대한 정보를 조회하고 정의된 작업의 내용을 수행한다

HPL (High Performance Loader)

- **Unload / load job 생성 및 실행**
  - **job 생성**
    - **ipload에서 job 생성**
    - **onpladm을 이용하여 job 생성**
      구문 :
      onpladm create job JOB이름
      –d DEVICE이름 –D DB명 –t TABLE이름 –f{ l | u }
  - **job 실행**
    - **ipload에서 job 실행**
    - **onpladm을 이용하여 job 실행**
      구문 :
      onpladm run job JOB이름 –f{ l | u }
    - **onpload 명령으로 직접 실행**
      구문 :
      onpload –j JOB이름 –f{ l | u }



ipload에서 job 자동 생성

- ipload에서 job을 생성할 때 HPL에 필요한 오브젝트(예,device, format, query, map, filter등)을 먼저 정의하는 방법이 있고 job generate 기능을 이용하여 관련된 오브젝트를 자동으로 생성하는 방법이 있다.
  - ipload의 auto generating
    - ipload 실행
    - Component메뉴의 generate job선택
    - Load 또는 Unload job 라디오 버튼 선택
    - format type 클릭
    - job 정보 입력
      - Generate name
      - Database
      - Table
      - Device
    - OK 클릭

- onpladm을 이용하여 job 생성하는 예

```
Informix> onpladm create job unload_job -d /data/data.txt
          -D stores_demo -t unload_table –fu
```

- onpladm또는 onpload를 사용하여 생성된 job을 실행

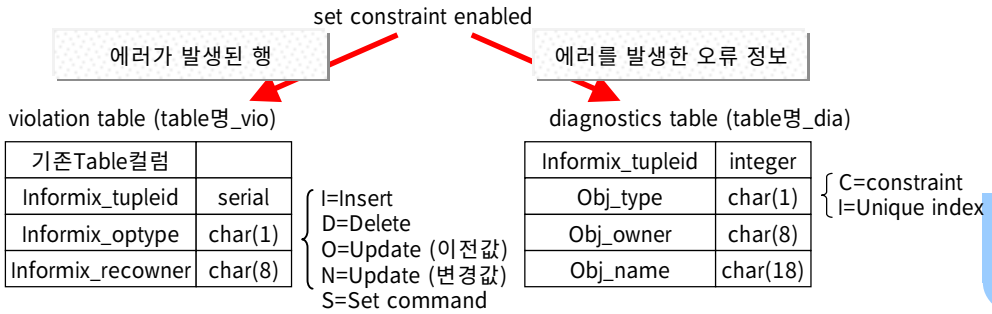
```
Informix> onpladm run job unload_job –fu
Informix> onpload -j unload_job –fu
```

- Tip :
  - device 오브젝트를 수정하여 여러 파일로 unload하거나 load하게 하면 HPL수행 속도가 더욱 빨라진다

HPL (High Performance Loader)

- **load job 모드**
  - **express**
    - **속도가 가장 빠르다**
    - **로드 중에는 constraint, index, trigger를 disable시킨다**
    - **로드 중에 테이블에 lock를 사용한다**
    - **standard 데이터 타입만을 지원한다**
    - **로드 후 0 레벨 백업을 받기 전까지는 읽기 전용 상태로 남는다**
    - **페이지 크기 보다 큰 row의 테이블은 로드 할 수 없다**
    - **ROWID를 사용한 fragment 테이블은 로드 할 수 없다**
  - **deluxe**
    - **constraint, index, trigger가 적용된 상태에서 로드**
    - **data replication 상태에서 로드 가능**
    - **로드 중에 다른 사용자가 그 테이블을 접근할 수 있다**
    - **모든 데이터 타입을 지원한다**

- 로드 작업 중 unique index, constraint등의 에러가 발생하거나(deluxe모드), disable된 unique index, constraint 등을 enable 시킬 때(express 모드) 에러가 발생하면 에러가 발생한 행에 대한 정보를 violation 테이블과 diagnostics 테이블에 저장한다



- Violation 테이블에 filtering된 행을 읽고 그 원인이 무엇인지 찾아서 violation 테이블을 수정하고 기존 테이블에 입력한다

```
update original_tab_vio set col1 = 123 where col1 = 100;
insert into orginal_tab
       select col1, col2, col3 from orginal_tab_vio
        where informix_tupleid = 1;
```

- 시스템 카타로그 테이블을 모니터링하여 현재 데이터베이스안에 어느 테이블의 constraint, index, trigger가 disable된 상태로 남아 있는지 알 수 있다

```
select tabname, objtype, state
  from systables a, sysobjstate b
 where a.tabid = b.tabid
```

C=Constraint
I=Index
T=Trigger

D=Disabled
E=Enabled
F=Filtering with no error
G=Filtering with error

HPL (High Performance Loader)

- **onpladm 기타 옵션**
  - **job 및 object 목록 조회**
    onpladm list { job | map | query | device | filter | format }
  - **job 및 object 내용 조회**
    onpladm desc job JOB이름 –f{ l | u } [–R]
    onpladm desc map MAP이름 –f{ l | u }
    onpladm desc { format | query | filter | device } 오브젝트이름
  - **job 및 object 삭제**
    onpladm delete job JOB이름 –f{ l | u } [–R]
    onpladm delete map MAP이름 –f{ l | u }
    onpladm delete { format | query | filter | device } 오브젝트이름
  - **서식파일을 통해 job 및 object 생성,수정**
    onpladm { create | modify } object –F 파일이름

- job 내용 조회 및 삭제 옵션 중 –R을 사용하면 해당 job과 관련된 모든 오브젝트(map,query,device,filter,format)를 모두 조회하거나 삭제할 수 있다
- 서식 파일을 통해 object를 수정할 수 있는데, 서식파일의 포맷을 정확히 모르면 기존에 생성된 object의 내용을 먼저 파일로 받아 수정하여 사용할 수 있다.
- unload file을 두 개로 바꾸는 예

```
Informix> onpladm desc device orders_unload > device.modify
Informix> vi device.modify
     1  BEGIN  OBJECT  DEVICEARRAY    orders_unload
     2
     3  BEGIN  SEQUENCE
     4  TYPE                    FILE
     5  FILE                    /tmp/orders.txt
     6  TAPEBLOCKSIZE           0
     7  TAPEDEVICESIZE          0
     8  PIPECOMMAND
     9  END  SEQUENCE
    10
    11  BEGIN  SEQUENCE
    12  TYPE                    FILE
    13  FILE                    /tmp/orders2.txt
    14  TAPEBLOCKSIZE           0
    15  TAPEDEVICESIZE          0
    16  PIPECOMMAND
    17  END  SEQUENCE
    18
    19  END  OBJECT
Informix > onpladm modify object -F device.modify
Successfully modified object DEVICEARRAY   orders_unload
```

위의 3~9 라인을 복사하여 붙여넣고 다른 이름의 파일로 수정한다

- Tip : Command로 제어할 수 없는 몇 가지 정보는 onpload 데이터베이스를 직접 수정하여 사용한다.
  - delimiter변경 : formats 테이블에서 fieldsep 컬럼 수정
  - 한글 로케일 사용 (ko_kr.ksc) : formats 테이블에서 datatype 컬럼 값을 ASCII에서 KS5601로 변경
  - run mode 변경 : session 테이블의 runmode 컬럼 수정
    - load : 129 →deluxe, 130 → express
    - unload : 129 → Dirty read, 130 → Committed Read