Physical Database Design and Referential Integrity

University of California, Berkeley

School of Information

INFO 257: Database Management

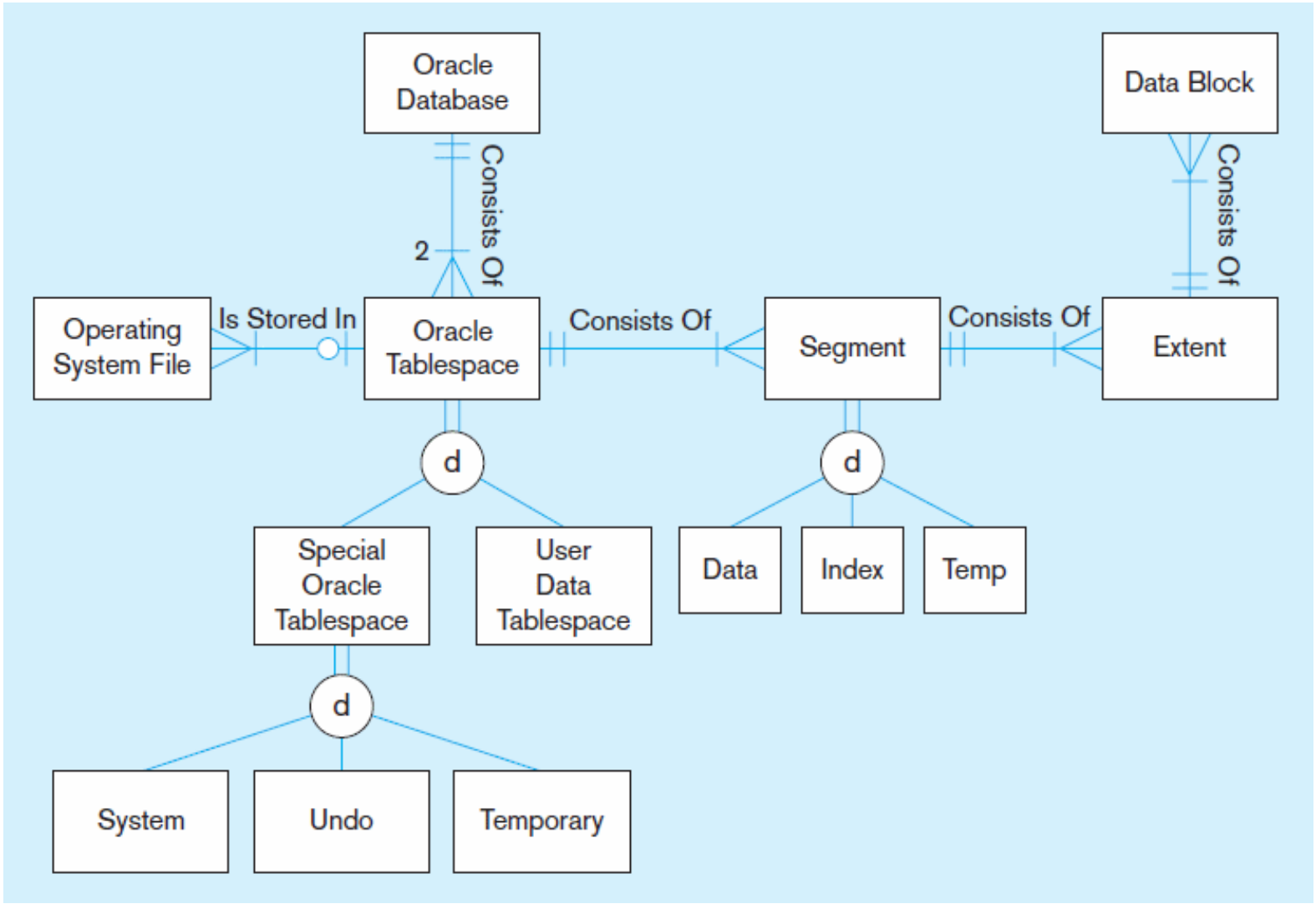
Lecture Outline

- File and Access Methods
- Indexes and What to index
- Integrity constraints
- Backups

Designing Physical Database Files

- Physical File:
  - A named portion of secondary memory allocated for the purpose of storing physical records
  - Tablespace–named logical storage unit in which data from multiple tables/views/objects can be stored
- Tablespace components
  - Segment – a table, index, or partition
  - Extent–contiguous section of disk space
  - Data block – smallest unit of storage

Figure 5-6 DBMS terminology in an Oracle 11g environment

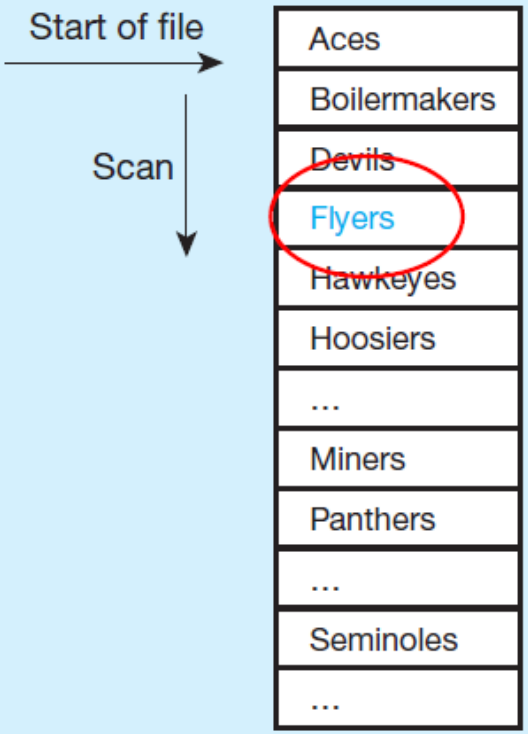


File Organizations

- Types of file organizations
  - Heap – no particular order
  - Sequential
  - Indexed
  - Hashed
- Factors for selecting file organization
  - Fast data retrieval and throughput
  - Efficient storage space utilization
  - Protection from failure and data loss
  - Minimizing need for reorganization
  - Accommodating growth
  - Security from unauthorized use

Comparison of File Organizations

a) Sequential

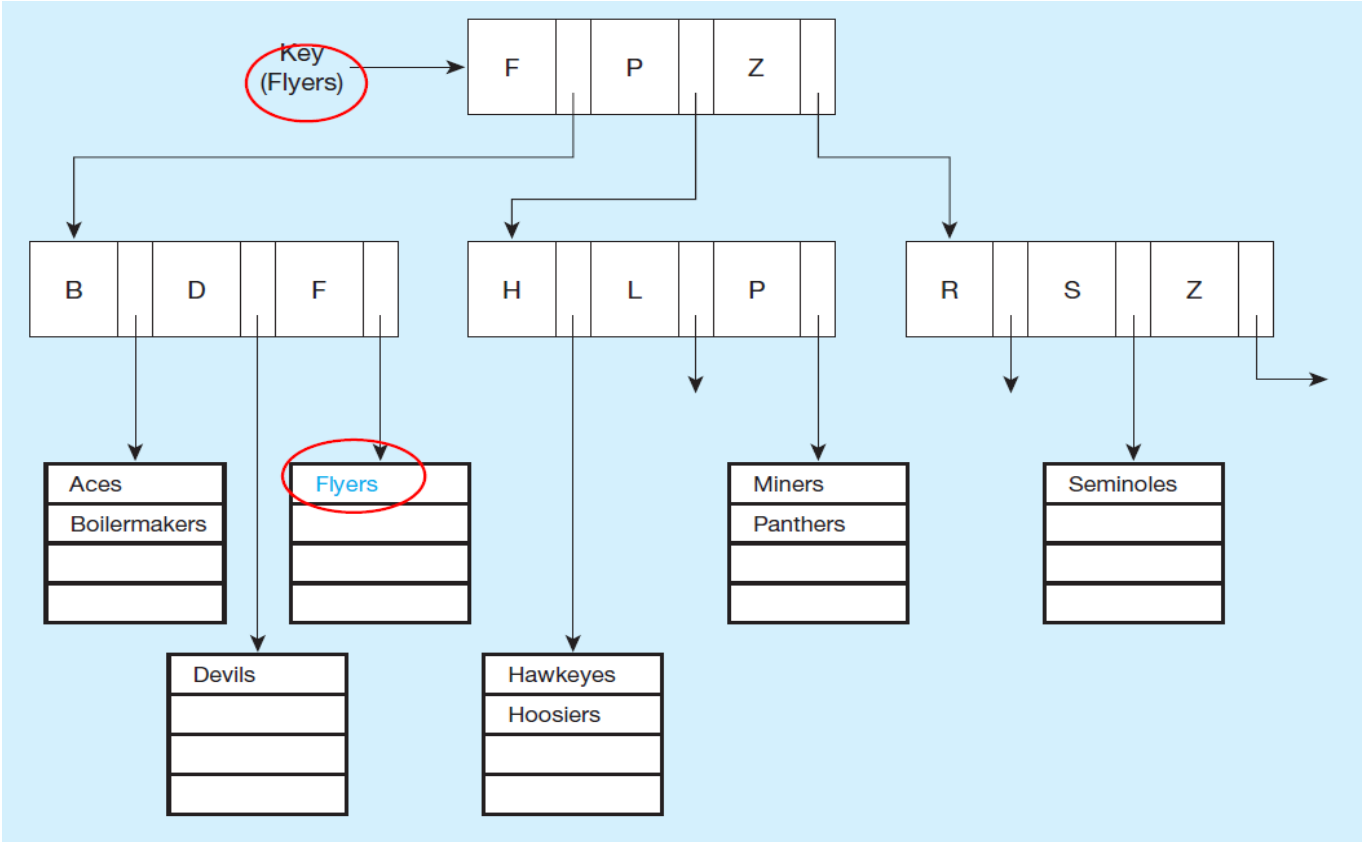


Indexed File Organizations

- Storage of records sequentially or nonsequentially with an index that allows software to locate individual records
- Index: a table or other data structure used to determine in a file the location of records that satisfy some condition
- Primary keys are automatically indexed
- Other fields or combinations of fields can also be indexed; these are called secondary keys (or nonunique keys)

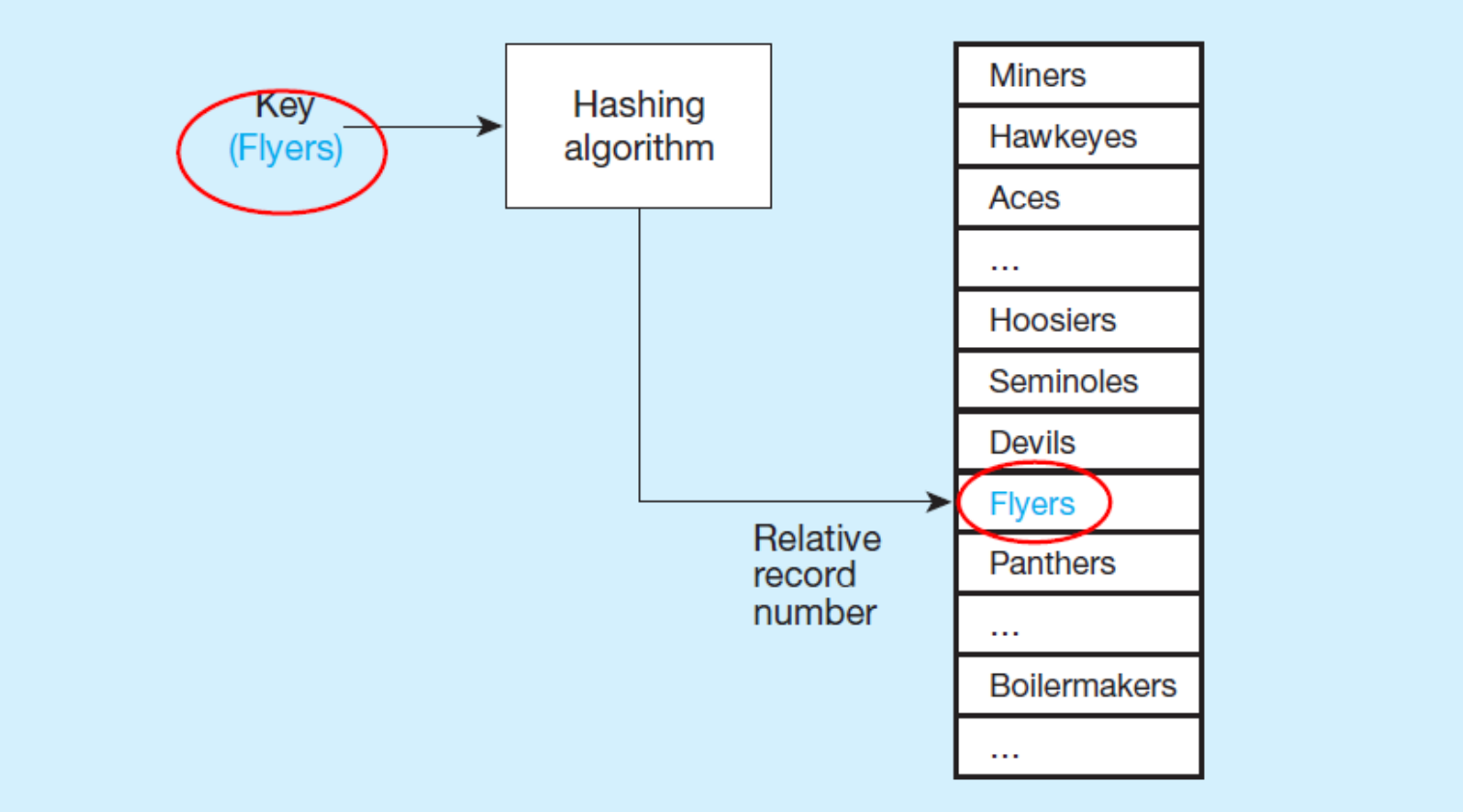
Comparison of File Organizations

a) Indexed



Comparison of File Organizations

a) Hashed



Clustering Files

- In some relational DBMSs, related records from different tables can be stored together in the same disk area
- Useful for improving performance of join operations
- Primary key records of the main table are stored adjacent to associated foreign key records of the dependent table
- e.g. Oracle has a CREATE CLUSTER command

Lecture Outline

- File and Access Methods
- Indexes and What to index
- Integrity constraints
- Backups

Indexes

- Most database applications require:
  - locating rows in tables that match some condition (e.g. SELECT operations)
  - Joining one table with another based on common values of attributes in each table
- Indexes can greatly speed up these processes and avoid having to do sequential scanning of database tables to resolve queries

Figure 6-8 **Join** Indexes–speeds up join operations

a) Join index for common non-key columns

Customer

RowID	Cust#	CustName	City	State
10001	C2027	Hadley	Dayton	Ohio
10002	C1026	Baines	Columbus	Ohio
10003	C0042	Ruskin	Columbus	Ohio
10004	C3861	Davies	Toledo	Ohio
. . .				

Store

RowID	Store#	City	Size	Manager
20001	S4266	Dayton	K2	E2166
20002	S2654	Columbus	K3	E0245
20003	S3789	Dayton	K4	E3330
20004	S1941	Toledo	K1	E0874
. . .				

Join Index

CustRowID	StoreRowID	Common Value*
10001	20001	Dayton
10001	20003	Dayton
10002	20002	Columbus
10003	20002	Columbus
10004	20004	Toledo
. . .		

*This column may or may not be included, as needed. Join index could be sorted on any of the three columns. Sometimes two join indexes are created, one as above and one with the two RowID columns reversed.

b) Join index for matching foreign key (FK) and primary key (PK)

Order

RowID	Order#	Order Date	Cust#(FK)
30001	O5532	10/01/2001	C3861
30002	O3478	10/01/2001	C1062
30003	O8734	10/02/2001	C1062
30004	O9845	10/02/2001	C2027
. . .			

Customer

RowID	Cust#(PK)	CustName	City	State
10001	C2027	Hadley	Dayton	Ohio
10002	C1062	Baines	Columbus	Ohio
10003	C0042	Ruskin	Columbus	Ohio
10004	C3861	Davies	Toledo	Ohio
. . .				

Join Index

CustRowID	OrderRowID	Cust#
10001	30004	C2027
10002	30002	C1062
10002	30003	C1062
10004	30001	C3861
. . .		

Type of Keys

- Primary keys -- as we have seen before -- uniquely identify a single row in a relational table
- Secondary keys -- are search keys that may occur multiple times in a table
- Bitmap Indexes
  - Table of bits where each row represents a distinct key value and each column is a bit – 0 or 1 for each record

Primary Key Indexes

- In MySQL – will create a unique index
- In Access – also this will be created automatically when a field is selected as primary key
  - in the table design view select an attribute row (or rows) and clock on the key symbol in the toolbar.
  - The index is created automatically as one with (No Duplicates)
- In SQL
    - CREATE UNIQUE INDEX indexname ON tablename(attribute);

Secondary Key Indexes

- In Access -- Secondary key indexes can be created on any field.
  - In the table design view, select the attribute to be indexed
  - In the "Indexed" box on the General field description information at the bottom of the window, select "Yes (Duplicates OK)"
- In SQL (including MySQL)
    - CREATE INDEX indxname on tablename(attribute);
- MySQL suggests that CREATE TABLE be used for most index creation. E.g. adding "*create table..,idnum int index using btree,*"

MySQL Index Creation syntax

```
CREATE [ONLINE|OFFLINE] [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name
    [index_type]
    ON tbl_name (index_col_name,...)
    [index_option] ...

index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH}

index_option:
    KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'
```

CREATE INDEX cannot be used to create a PRIMARY KEY; use ALTER TABLE instead

When to Index

- Tradeoff between time and space:
  - Indexes permit faster processing for searching
  - But they take up space for the index
  - They also slow processing for insertions, deletions, and updates, because both the table and the index must be modified
- Thus they **SHOULD** be used for databases where search is the main mode of interaction
- The might be skipped if high rates of updating and insertions are expected, and access or retrieval operations are rare

When to Use Indexes

- Rules of thumb
  - Indexes are most useful on larger tables
  - Specify a unique index for the primary key of each table (automatically done for many DBMS)
  - Indexes are most useful for attributes used as search criteria or for joining tables
  - Indexes are useful if *sorting* is often done on the attribute
  - Most useful when there are many different values for an attribute
  - Some DBMS limit the number of indexes and the size of the index key values
  - Some indexes will not retrieve NULL values

Query Optimization

- Parallel query processing–possible when working in multiprocessor systems
- Overriding automatic query optimization–allows for query writers to preempt the automated optimization
- Oracle example:

```
SELECT /*+ FULL(Order_T) PARALLEL(Order_T,3) */ COUNT(*)
FROM Order_T
WHERE Salesperson = "Smith";
```

/* */ clause is a hint to override Oracle's default query plan

Data Dictionaries and Repositories

- Data dictionary
  - Documents data elements of a database
- System catalog
  - System-created database that describes all database objects
- Information Repository
  - Stores metadata describing data and data processing resources

Lecture Outline

- File and Access Methods
- Indexes and What to index
- Integrity constraints
- Backups

Views and Integrity Controls

- Views
  - Subset of the database that is presented to one or more users
  - User can be given access privilege to view without allowing access privilege to underlying tables
- Integrity Controls
  - Protect data from unauthorized use
  - Domains – set allowable values
  - Assertions – enforce database conditions
  - Triggers – prevent inappropriate actions, invoke special handling procedures, write to log files

Integrity Constraints

- The constraints we wish to impose in order to protect the database from becoming inconsistent.
- Five types
  - Required data
  - attribute domain constraints
  - entity integrity
  - referential integrity
  - enterprise constraints

Integrity Constraints

- The constraints we wish to impose in order to protect the database from becoming inconsistent.
- Five types
  - Required data
  - attribute domain constraints
  - entity integrity
  - referential integrity
  - enterprise constraints

Integrity constraints

- Usually set during table creation in RDBMS
- May also be set or modified by ALTER TABLE

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] *tbl_name* (*create_definition*,...) [*table_options*]

Required Data

- Some attributes must always contain a value -- they cannot have a null
- For example:
  - Every employee must have a job title.
  - Every diveshop diveitem must have an order number and an item number.

Attribute Domain Constraints

- Every attribute has a domain, that is a set of values that are legal for it to use
- For example:
  - The domain of sex in the employee relation is "M" or "F"
- Domain ranges can be used to validate input to the database.

E.g. – in SQLite

- sqlite> CREATE TABLE tst (num integer CHECK (num < 100));
- sqlite> insert into tst (num) values (1);
- sqlite> select * from tst;
- 1
- sqlite> insert into tst (num) values (80);
- sqlite> insert into tst (num) values (99);
- sqlite> insert into tst (num) values (100);
- Error: constraint failed

Entity Integrity

- The primary key of any entity cannot be NULL.

Column Definitions in MySQL

- *column_definition*:

  data_type [NOT NULL | NULL]

  [DEFAULT *default_value*]
  [AUTO_INCREMENT]

  [UNIQUE [KEY] | [PRIMARY] KEY]

  [COMMENT '*string*']

  [COLUMN_FORMAT
  {FIXED|DYNAMIC|DEFAULT}]

  [STORAGE {DISK|MEMORY|DEFAULT}]
  [*reference_definition*]

Referential Integrity

- A "foreign key" links each occurrence in a relation representing a *child* entity to the occurrence of the *parent* entity containing the matching candidate key
- Referential Integrity means that if the foreign key contains a value, that value *must refer to an existing occurrence* in the parent entity
- For example:
  - Since the 'Order ID' in the diveitem relation refers to a particular diveords primary key, that key –and row- must exist for referential integrity to be satisfied

Referential Integrity

- Referential integrity options are declared when tables are defined (in most systems)
- There are many issues having to do with how particular referential integrity constraints are to be implemented to deal with insertions and deletions of data from the parent and child tables.

Insertion rules

- A row should not be inserted in the referencing (child) table unless there already exists a matching entry in the referenced table.
- Inserting into the parent table should *not* cause referential integrity problems
  - Unless it is itself a child…
- Sometimes a special NULL value may be used to create child entries without a parent or with a "dummy" parent.

Deletion rules

- A row should not be deleted from the referenced table (parent) if there are matching rows in the referencing table (child).
- Three ways to handle this
  - **Restrict** -- disallow the delete
  - **Nullify** -- reset the foreign keys in the child to some NULL or dummy value
  - **Cascade** -- Delete all rows in the child where there is a foreign key matching the key in the parent row being deleted

E.g. – in MySQL

- *reference_definition*:

  REFERENCES *tbl_name* (*index_col_name*,...)
[MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]

  [ON DELETE *reference_option*]

  [ON UPDATE *reference_option*]

- *reference_option*:

  RESTRICT | CASCADE | SET NULL | NO ACTION

Referential Integrity

- This can be implemented using external programs that access the database
- newer databases implement executable rules or built-in integrity constraints

Authorization Rules

- Controls incorporated in the data management system
- Restrict:
  - access to data
  - actions that people can take on data
- Authorization matrix for:
  - Subjects
  - Objects
  - Actions
  - Constraints

Subject	Object	Action	Constraint
Sales Dept.	Customer record	Insert	Credit limit LE $5000
Order trans.	Customer record	Read	None
Terminal 12	Customer record	Modify	Balance due only
Acctg. Dept.	Order record	Delete	None
Ann Walker	Order record	Insert	Order aml LT $2000
Program AR4	Order record	Modify	None

Implementing Authorization Rules

a) Authorization table for subjects (salespersons)

	Customer records	**Order records**
Read	Y	Y
Insert	Y	Y
Modify	Y	N
Delete	N	N

b) Authorization table for objects (orders)

	Salespersons (password BATMAN)	**Order entry (password JOKER)**	**Accounting (password TRACY)**
Read	Y	Y	Y
Insert	N	Y	N
Modify	N	Y	Y
Delete	N	N	Y

DIVEORDS SQL

```
CREATE TABLE `DIVEORDS` (
`Order_No` int(11) NOT NULL,
`Customer_No` int(11) default NULL,
`Sale_Date` datetime default NULL,
`Ship_Via` varchar(255) default NULL,
`Ship_Cost` double default NULL,
…some things deleted for space…,
`VacationCost` double default NULL,
PRIMARY KEY (`Order_No`),
KEY `Customer_No` (`Customer_No`),
KEY `DESTDIVEORDS` (`Destination`),
KEY `DIVECUSTDIVEORDS` (`Customer_No`),
KEY `DIVEORDSShip_Via` (`Ship_Via`),
KEY `SHIPVIADIVEORDS` (`Ship_Via`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

DIVEITEM SQL

```
CREATE TABLE `DIVEITEM` (
`Order_No` int(11) NOT NULL,
`Item_No` int(11) default NULL,
 `Rental_Sale` varchar(255) default NULL,
`Qty` smallint(6) default NULL,
`Line_Note` varchar(255) default NULL,
KEY `DIVEORDSDIVEITEM` (`Order_No`),
KEY `DIVESTOKDIVEITEM` (`Item_No`),
KEY `Item_No` (`Item_No`),
FOREIGN KEY (`Order_No`) REFERENCES
        DIVEORDS(`Order_No`) ON DELETE CASCADE )
ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Note that only the InnoDB or NDB Engines in MySQL support actual actions and checking for Foreign Keys

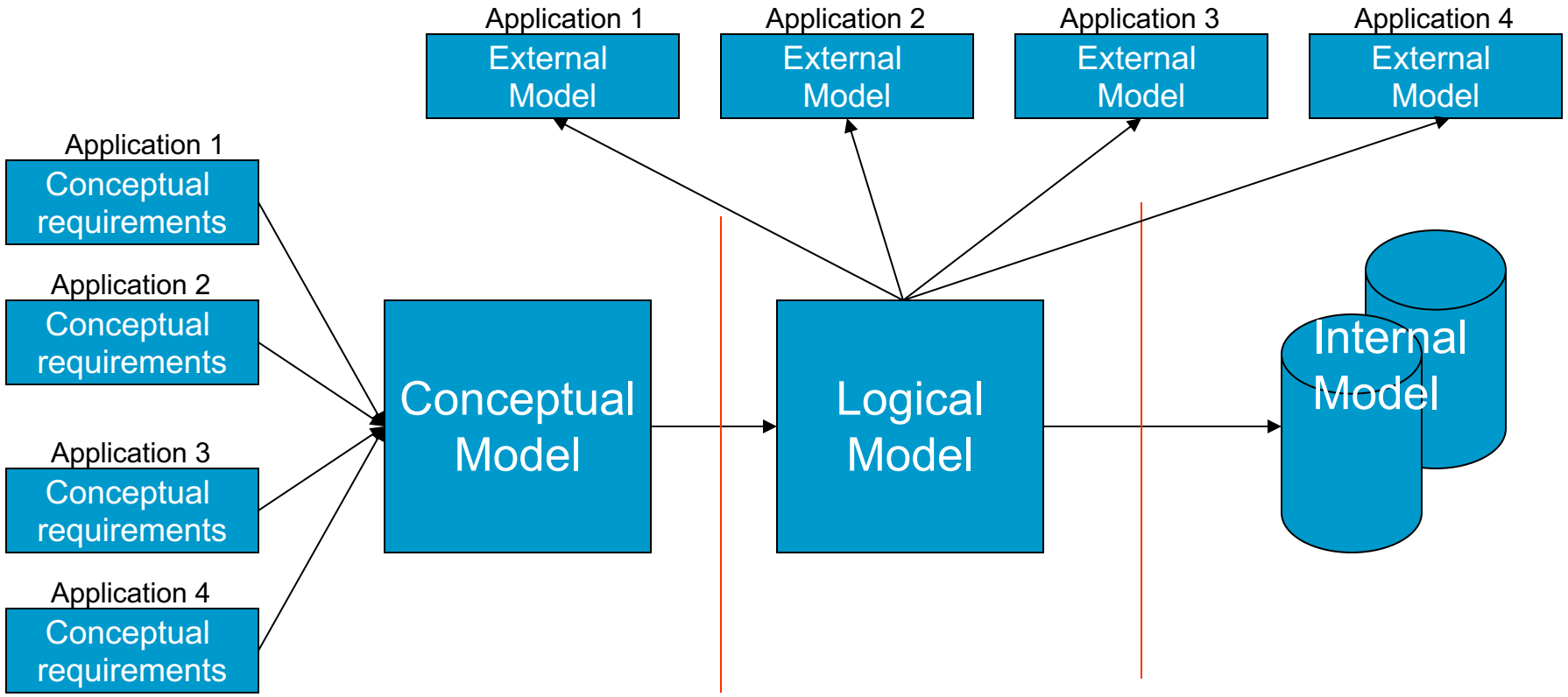
Enterprise Constraints

- These are business rule that may affect the database and the data in it
  - for example, if a manager is only permitted to manage 10 employees then it would violate an enterprise constraint to manage more

Lecture Outline

- File and Access Methods
- Indexes and What to index
- Integrity constraints
- Backups

Database Design Process



MySQL Backup Types

- Physical (Raw) Versus Logical Backups
  - Physical (or Raw) Backups
    - Physical backups consist of raw copies of the directories and files that store database contents. This type of backup is suitable for large, important databases that need to be recovered quickly when problems occur.
  - Logical Backups
    - Logical backups save information represented as logical database structure (CREATE DATABASE, CREATE TABLE statements) and content (INSERT statements or delimited-text files). This type of backup is suitable for smaller amounts of data where you might edit the data values or table structure, or recreate the data on a different machine architecture.

From: http://dev.mysql.com/doc/refman/5.1/en/backup-types.html

Logical Backup

- The backup is done by querying the MySQL server to obtain database structure and content information.
- Backup is slower than physical methods because the server must access database information and convert it to logical format.
- Output is larger than for physical backup, particularly when saved in text format.
- Backup and restore granularity is available at the server level (all databases), database level (all tables in a particular database), or table level. This is true regardless of storage engine.
- The backup does not include log or configuration files, or other database-related files that are not part of databases.
- Backups stored in logical format are machine independent and highly portable.
- Logical backups are performed with the MySQL server running.

Logical Backups

- Logical backup tools include the **mysqldump** program and the **SELECT ... INTO OUTFILE** statement. These work for any storage engine, even MEMORY.
- To restore logical backups, SQL-format dump files can be processed using the **mysql** client. To load delimited-text files, use the **LOAD DATA INFILE** statement or the **mysqlimport** client.

Logical Backups

- mysqldump –p [-X] *databasename tablename(s)*
- *Demo of normal and XML output*

Physical Backups

- The backup consists of exact copies of database directories and files. Typically this is a copy of all or part of the MySQL data directory.
- Physical backup methods are faster than logical because they involve only file copying without conversion.
- Output is more compact than for logical backup.
- Backup and restore granularity ranges from the level of the entire data directory down to the level of individual files.
- In addition to databases, the backup can include any related files such as log or configuration files.
- Backups are portable only to other machines that have identical or similar hardware characteristics.
- Backups can be performed while the MySQL server is not running. If the server is running, it is necessary to perform appropriate locking so that the server does not change database contents during the backup.

Physical Backups

- Physical backup tools include file system-level commands (such as cp, scp, tar, rsync), **mysqlhotcopy** for MyISAM tables, **ibbackup** for InnoDB tables, or **START BACKUP** for NDB tables.
- For restore, files copied at the file system level or with **mysqlhotcopy** can be copied back to their original locations with file system commands; **ibbackup** restores InnoDB tables, and **ndb_restore** restores NDB tables.