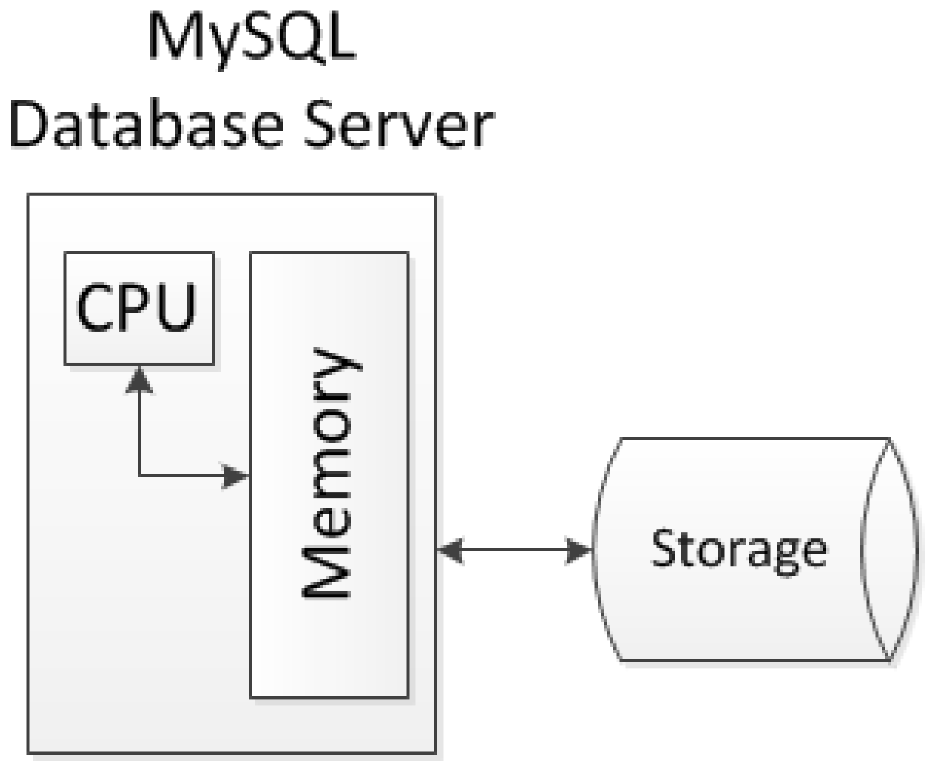
DBMS Indexes, Backup and Transactions

The concept of an index

- Think of the index at the back of a book
- Speeds up a search
  - Look up a key word in the index
  - Index points to a page number in the book
- Without the index, you would have to scan every page in the book looking for your key word

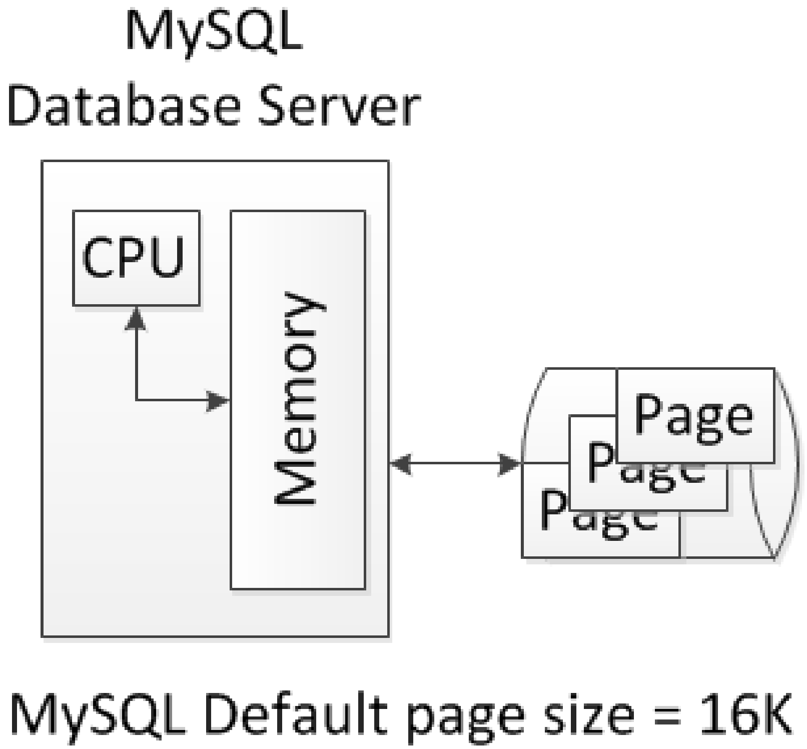
Consider database server architecture



Relative Speeds

nanoseconds
microseconds
milliseconds

DBMS Indexes



Data is moved from disk to memory one page at a time.

DBMS Indexes

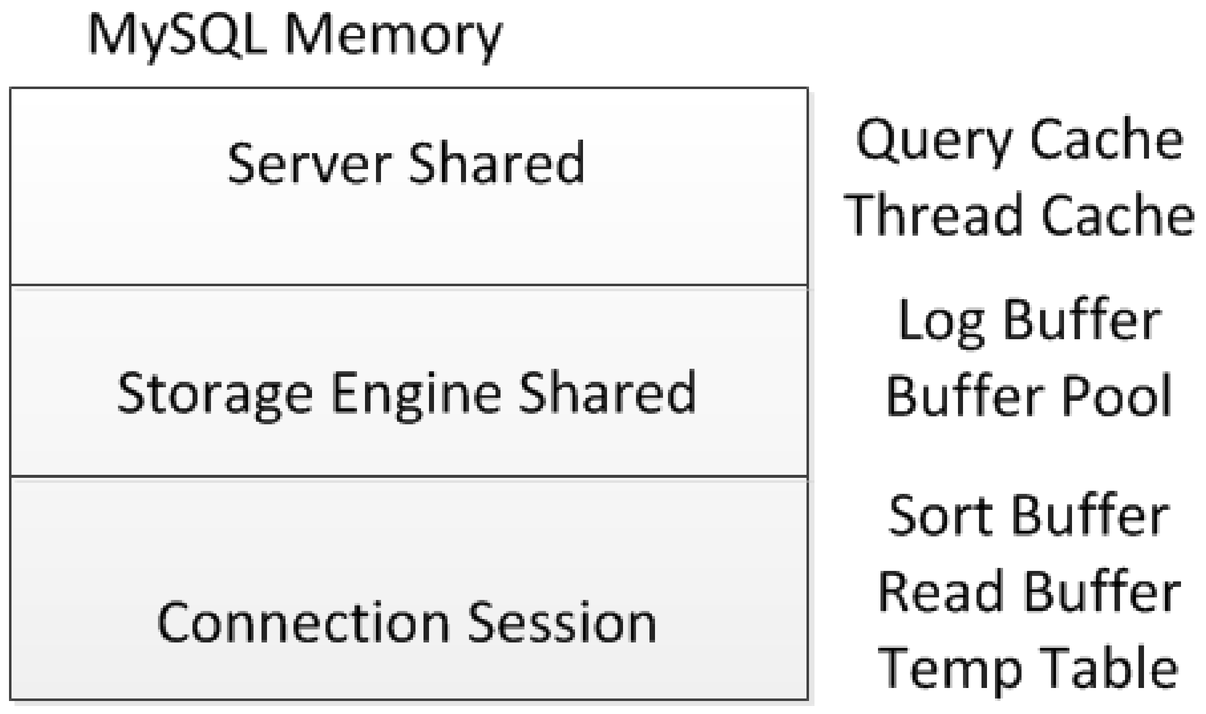
To read an entire table, the table is moved into memory one page at a time.

Any select, insert, update, delete requires the affected page where that row resides to be brought into memory (“buffer cache.”)

DML statements modify affected rows on the page in memory. Modified pages are written to disk upon a “commit”.

Moving pages into and out of memory is relatively very time consuming.

DBMS Indexes



DBMS Indexes

MySQL memory allocation is divided into two categories:

- Global (per instance)

Allocated when the instance starts and freed when the instance is shut down. If it fills up, the OS will swap, DB performance will suffer, and eventually the server will crash.

- Session (per connection)

Allocated per session. Released at session end. Used for query results. Read buffer size is per session. 10MB read buffer * 100 concurrent sessions = 1GB of memory.

Query Cache

- Holds the compiled executable query.
- Queries may be run over and over by an application, changing only host variables.
  - Think of a drop-down list created at web page load that reads many rows.
  - Caching the query avoids the time spent validating, parsing, and compiling the query.
- Deprecated in MySQL 5.7
  - Reduces performance predicatability
  - Doesn’t scale with huge workloads on muli-core machines

Thread Cache

- Splits up the execution paths of MySQL into multiple threads, one per user connection, so they run in parallel.

Buffer Pools

- Caching disk I/O
- Can hold objects in memory (smaller tables, indexes)

Session Memory

- Smaller temp tables
- Client-specific memory
  - Table read buffers
  - Sort operations

DBMS Indexes

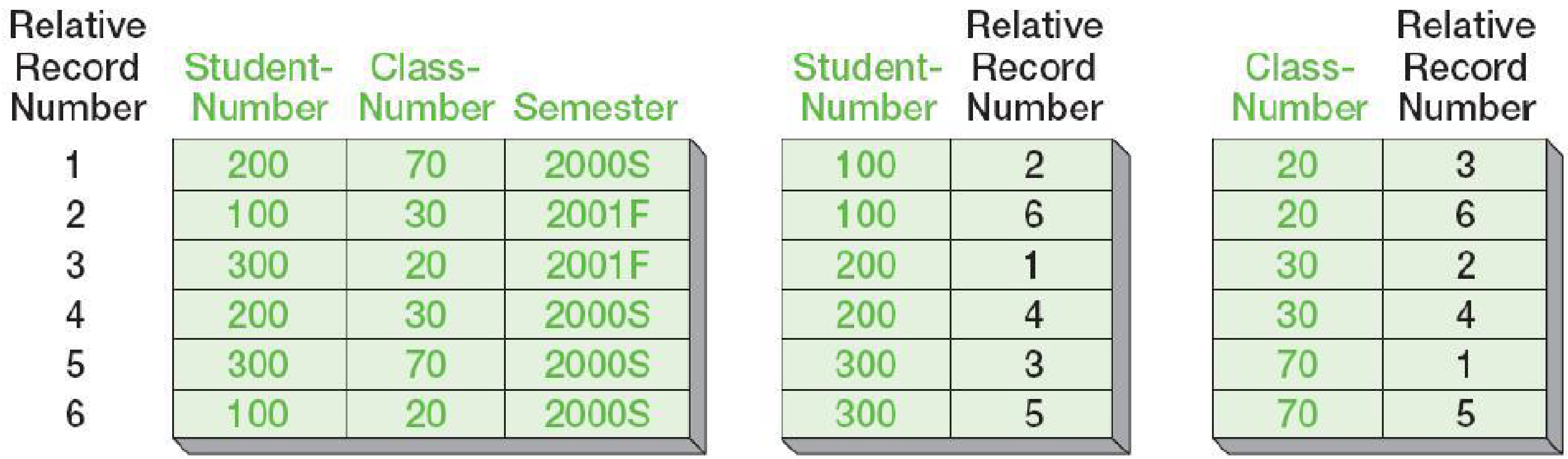
So why indexes?

We use indexes to minimize page movement (i.e. Minimize “disk I/O”).

Indexes are usually much smaller than the base tables they serve -- less I/O, less buffer space consumed.

Indexes are often pinned in memory.

Simple index structure.



Relative Record Number	Student-Number	Class-Number	Semester
1	200	70	2000S
2	100	30	2001F
3	300	20	2001F
4	200	30	2000S
5	300	70	2000S
6	100	20	2000S

(a) ENROLLMENT Data

Student-Number	Relative Record Number
100	2
100	6
200	1
200	4
300	3
300	5

(b) Index on StudentNumber

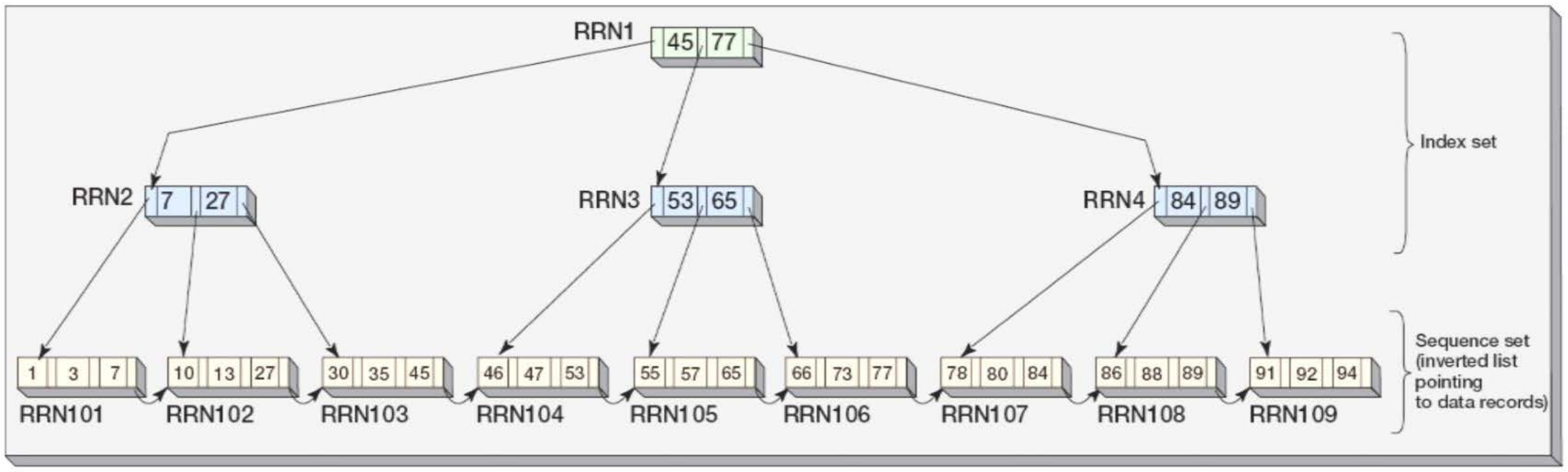
Class-Number	Relative Record Number
20	3
20	6
30	2
30	4
70	1
70	5

(c) Index on ClassNumber

MySQL indexes are typically B-Tree, the default.

- B-Tree supports both sequential access of rows AND keyed access.
- “Index Set and Sequence Set.”
- “root” and “leaf” nodes.

General Structure of a Simple B-Tree



DBMS Indexes

- The “B” means “balanced”
- B-Tree indexes balance each leaf page node between half-full and full.
- B-Tree indexes balance the levels of leaf pages.
- As pages get full, B-Tree indexes will “split” to create more room for indexes to grow. The index has pre-allocated overflow space.

DBMS Indexes

- As the overflow space gets full, the “split” leaf nodes get spead out.

- Therefore B-Tree indexes occasionally need to be reorganized.

- Drop the index. Recreate it.

DBMS Indexes

- The default DB engine for MySQL is “innodb”.

- A database or table can be created to use the “Memory” DB engine. All objects are memory resident, asynchronously written to disk.

- The Memory engine supports “hash” indexes. The key is run through a hashing algorithm which calculates a shorter key length, which is then associated with a pointer.

DBMS Indexes

- **Primary keys** are indexed by default.
- Data rows are written to disk in primary key sequence. This is a “clustered” index.
- Allows for binary search.

DBMS Indexes

- Foreign keys may be indexed.
- Other non-key columns may be indexed. ("Secondary" index.)
- The data rows will not be in secondary key sequence.
- MySQL secondary indexes include the primary key values so that access by secondary keys can leverage the clustered index.

DBMS Indexes

- Indexes present a trade-off:
  - READs go faster when indexed
  - INSERTs require not only data table updates, but also index updates.
  - For applications that are insert-intensive, indexes can cause significant delays.
  - For small tables, an index can actually hurt performance

DBMS Indexes

- SO –
  - Be very careful about creating indexes
  - Only create a new index when query performance demands it

DBMS Indexes

- **“Ad Hoc”** query access.
  - End-User Reporting tools generate SQL
    - QlikView
    - Tableau
    - Business Objects
  - Queries must be monitored for performance
  - Performance bottlenecks must be managed
  - Requires 3rd party tools.
    - WebYog, DataDog for monitoring
    - Queuing - Not in MySQL, but other DBMSs have it

DBMS Indexes

- For example:

```
CREATE TABLE test (
     id INT NOT NULL,
     last_name CHAR(30) NOT NULL,
     first_name CHAR(30) NOT NULL,
     PRIMARY KEY (id),
     INDEX IX_Name (last_name,first_name)
);
```

DBMS Indexes

Query use INDEX `IX_Name (last_name,first_name)`

```
SELECT * FROM test
WHERE last_name='Widenius';

SELECT * FROM test
WHERE last_name='Widenius' AND first_name='Michael';

SELECT * FROM test
WHERE last_name='Widenius'
AND (first_name='Michael' OR first_name='Monty');
```

DBMS Indexes

- Indexes can be included in the TABLE CREATE

- Indexes can be dropped, renamed, added later via a ALTER TABLE

```
ALTER TABLE contacts
    RENAME INDEX contacts_idx
    TO  contacts_new_index;

DROP INDEX contacts_idx
    ON contacts;
```

DBMS Indexes

MySQL keeps statistics on indexes in the “statistics” table in the “information_schema” table. And, innodb keeps table and index stats in the mysql schema.

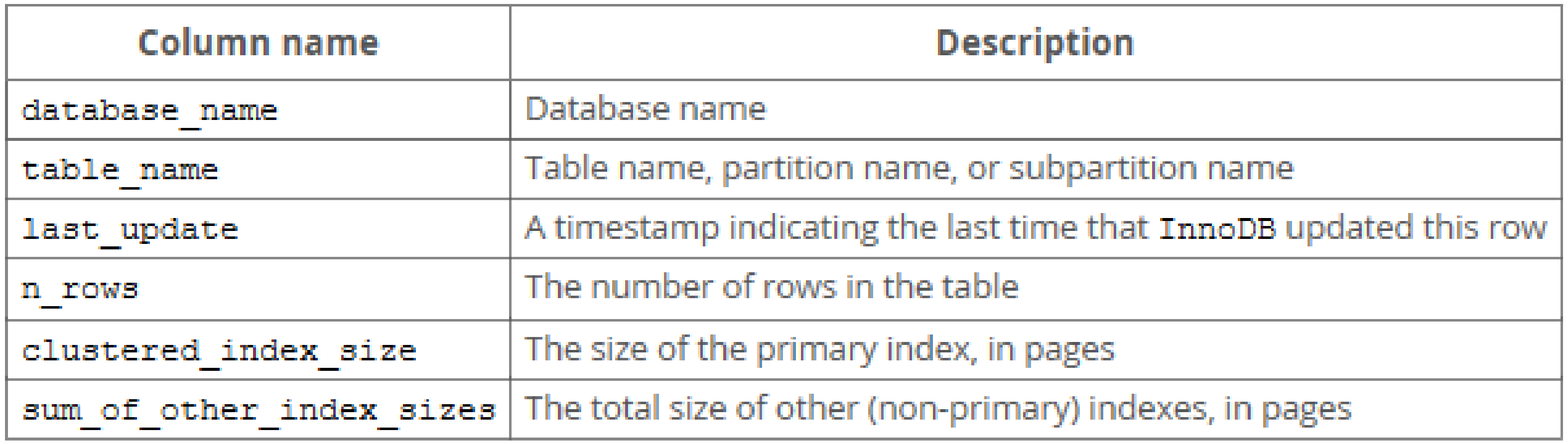
The query optimizer uses these statistics as it parses a query and builds the execution plan to decide whether or not to use an index.

If the statistics are “stale”, the optimizer might decide to NOT use an important index.

Therefore, statistics need to be refreshed.

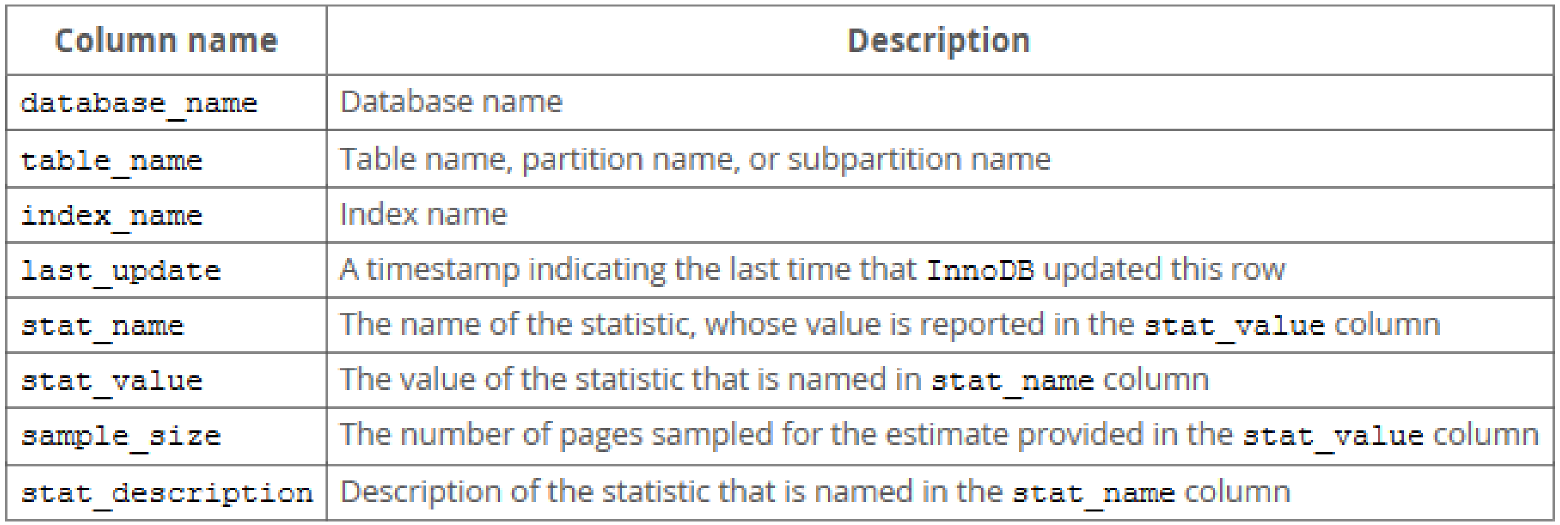
DBMS Indexes

Table 14.3 Columns of innodb_table_stats



Column name	Description
`database_name`	Database name
`table_name`	Table name, partition name, or subpartition name
`last_update`	A timestamp indicating the last time that `InnoDB` updated this row
`n_rows`	The number of rows in the table
`clustered_index_size`	The size of the primary index, in pages
`sum_of_other_index_sizes`	The total size of other (non-primary) indexes, in pages

Table 14.4 Columns of innodb_index_stats



Column name	Description
`database_name`	Database name
`table_name`	Table name, partition name, or subpartition name
`index_name`	Index name
`last_update`	A timestamp indicating the last time that `InnoDB` updated this row
`stat_name`	The name of the statistic, whose value is reported in the `stat_value` column
`stat_value`	The value of the statistic that is named in `stat_name` column
`sample_size`	The number of pages sampled for the estimate provided in the `stat_value` column
`stat_description`	Description of the statistic that is named in the `stat_name` column

DBMS Backup

DB BACKUP:

The DBAs' "bottom line" – never, ever lose data

So we run backups. How often?

– It depends on the size of the database, the number of users, the frequency of updates, and how critical the database is to the organization.

Some systems are SO CRITICAL that we keep redundant copies up-to-date at all times. Lose one, use the other.

- technology called **REPLICATION.**

Data Replication is the process of storing data in more than one node. It is simply copying data from a database from one server to another server.

DBMS Backup

Backups and Replication both rely on Transaction Logs

Every activity against the database that changes the database is written to a log file. Called “Binary Logs” or “BinLogs” in MySQL.

Log files reside in memory and are written to disk periodically. (In MySQL, this is called a “flush”.)

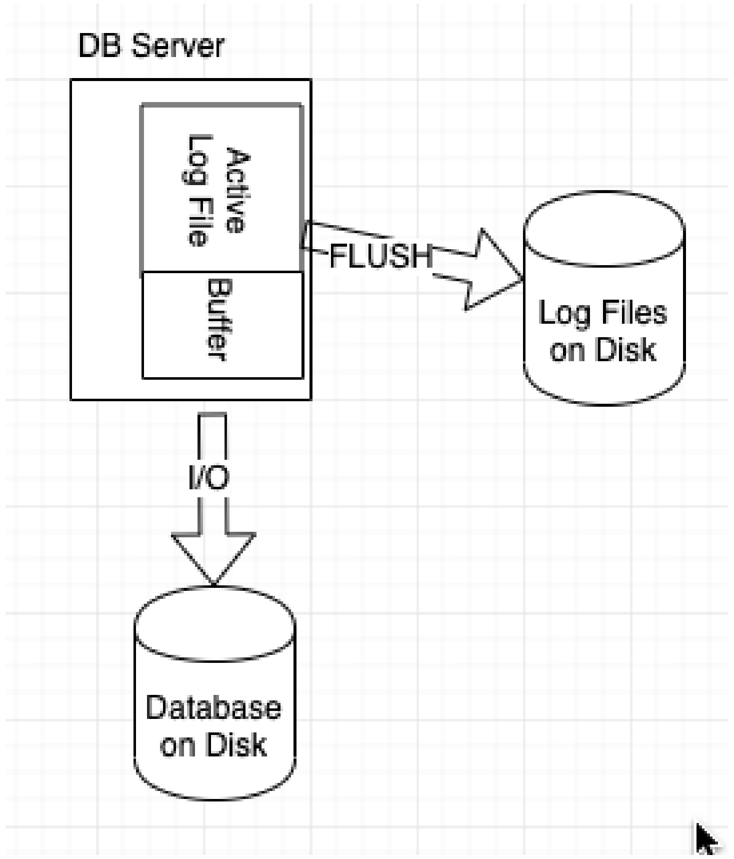
Transactions are written to the log file first, then written to the database.   (Write to log is sequential and faster.)

DBMS Backup

- **Logs in memory must be written to disk**
- **A log flush in MySQL**
- **(“redo logs” ➔ “archive logs” in Oracle.)**

- **Log files can be automatically written to disk**
  – On a timer, every nn minutes
  – On a size limit, whenever the log file grows to nn MB or GB
- **Closes out the old log file and opens up a new one**
  – Increments the log number in the filename

DBMS Backup



A database buffer is a temporary storage area in memory used to hold a copy of a database block while it is being moved from one place to another.

DBMS Backup

Types of backups – usually we combine all of these

1. **Stop all database traffic and back it up quickly**
   - “Cold” backup (“full” or “incremental”)
   - MySQLDUMP versus a file-level backup (OS level)
   - Faster
   - Requires downtime
2. **Take the backup while database traffic is in-progress**
   - “Warm” backup (“full” or “incremental”)
   - Takes longer
   - Less inconvenience to user community

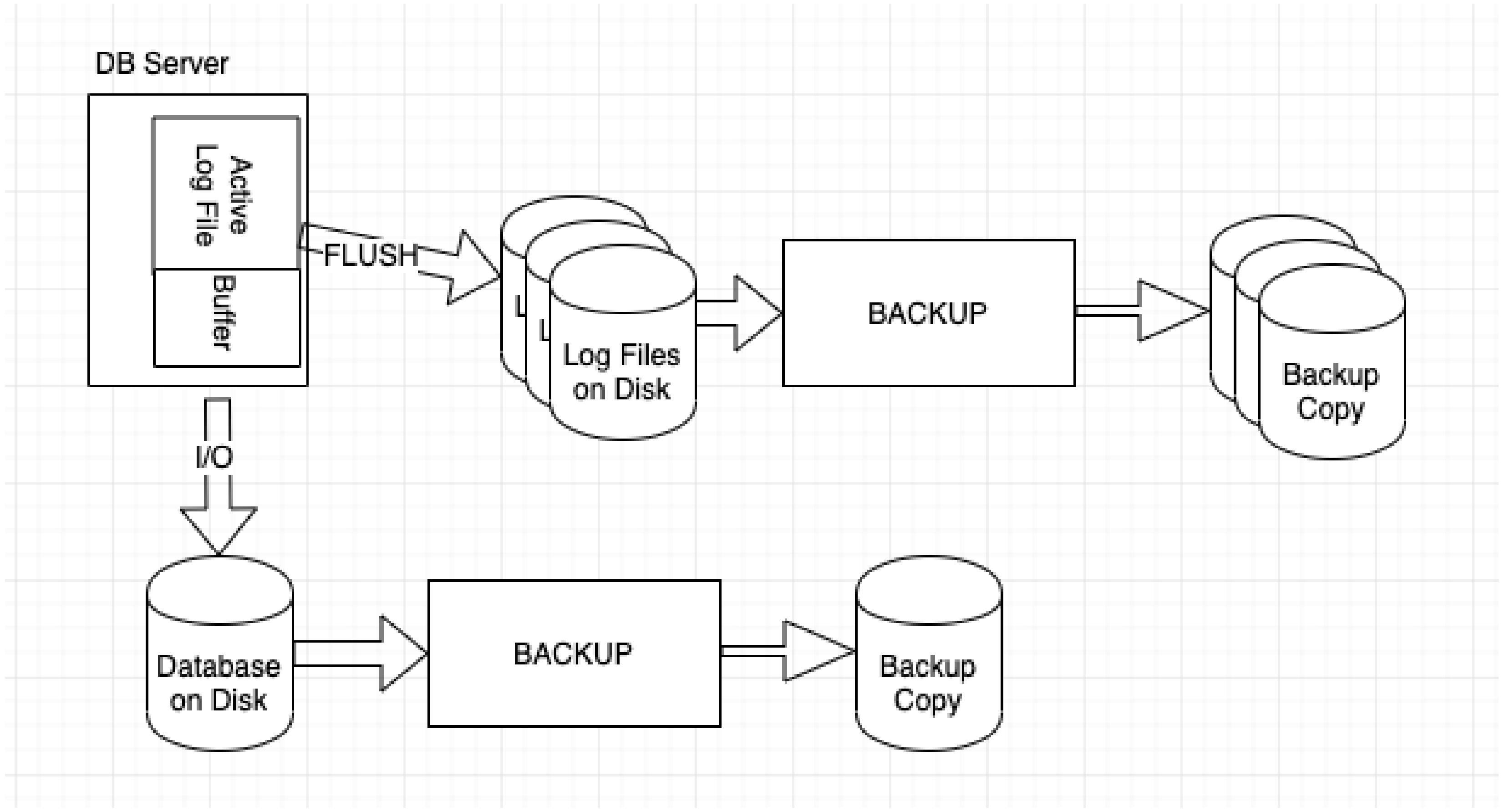
DBMS Backup

3. **Keep a duplicate up-to-date at all times**
   - “Hot standby” via **replication** from logs

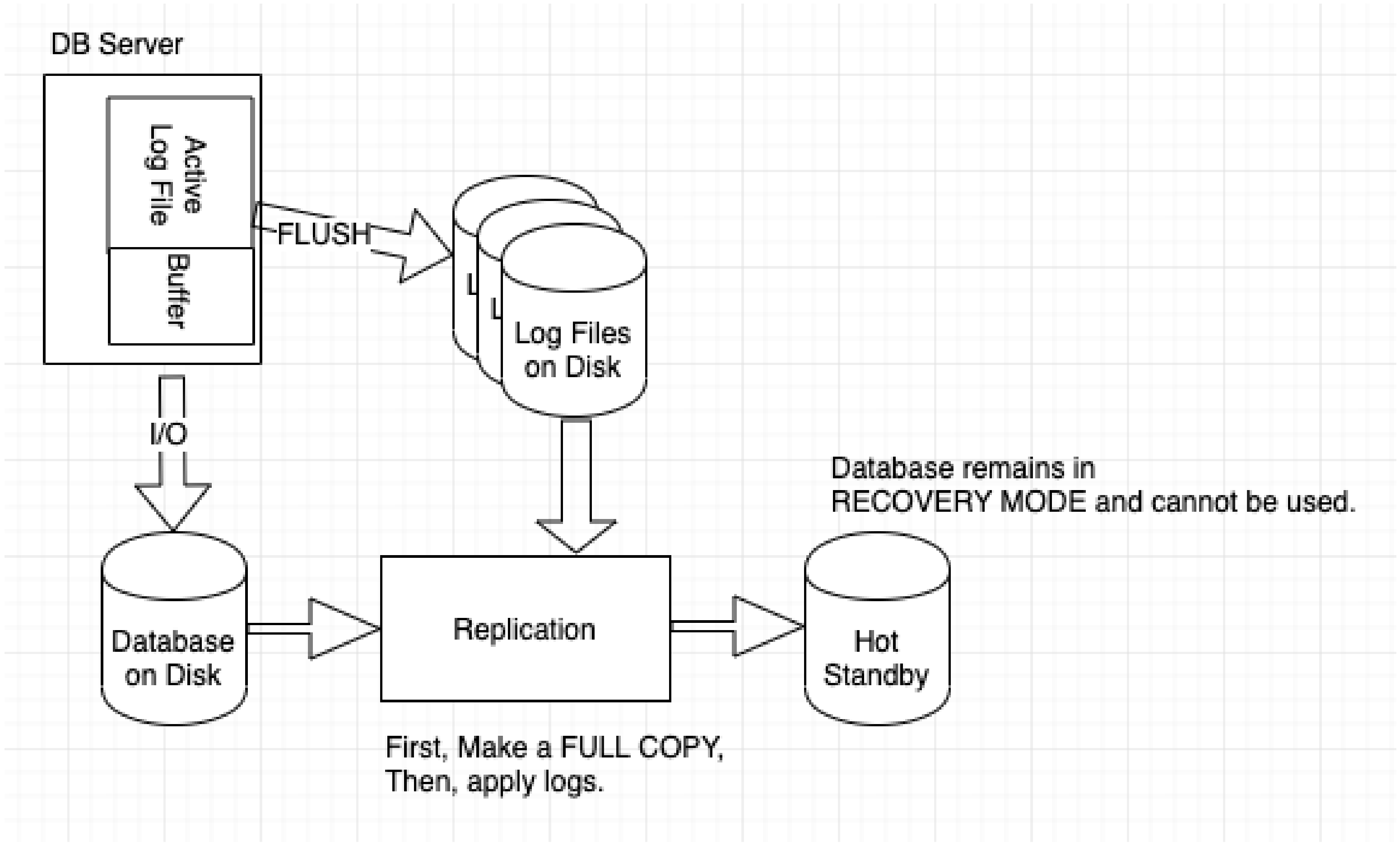
4. **Full versus Incremental**
   - The backup software copies the entire database (“full”) versus copying only rows changed since the last “full” (“incremental”)

DBMS Backup

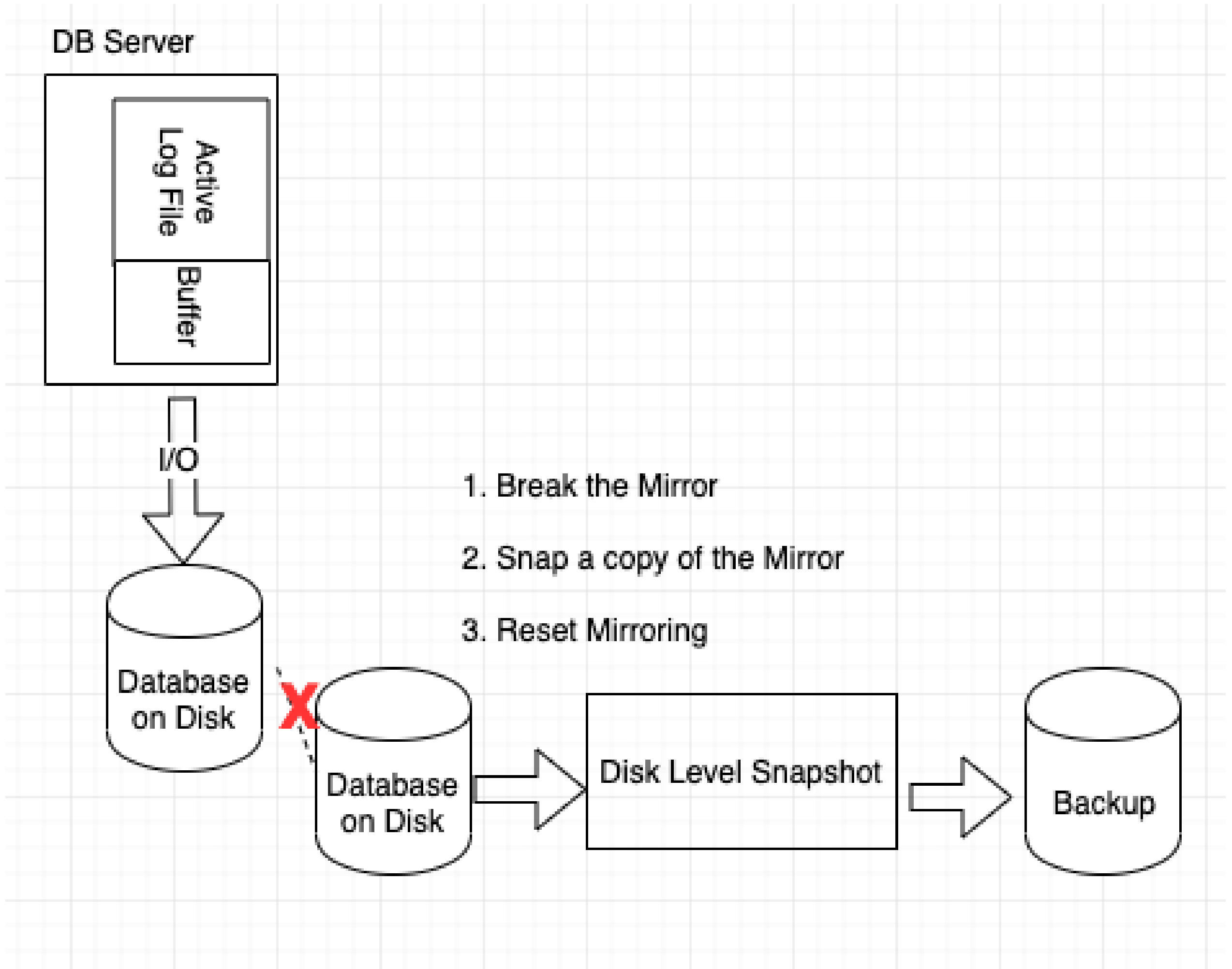
Standard Backup:



Hot Standby (via Replication):


DB Server
Active Log File
Buffer
FLUSH
Log Files on Disk
I/O
Database on Disk
Replication
Hot Standby
Database remains in RECOVERY MODE and cannot be used.
First, Make a FULL COPY, Then, apply logs.

Disk-Level Backup:



DBMS Backup

Logs can be used to UNDO a transaction or REDO a transaction.

Logs capture an image of the row BEFORE and AFTER it was changed.

Logs capture timestamps for every activity.

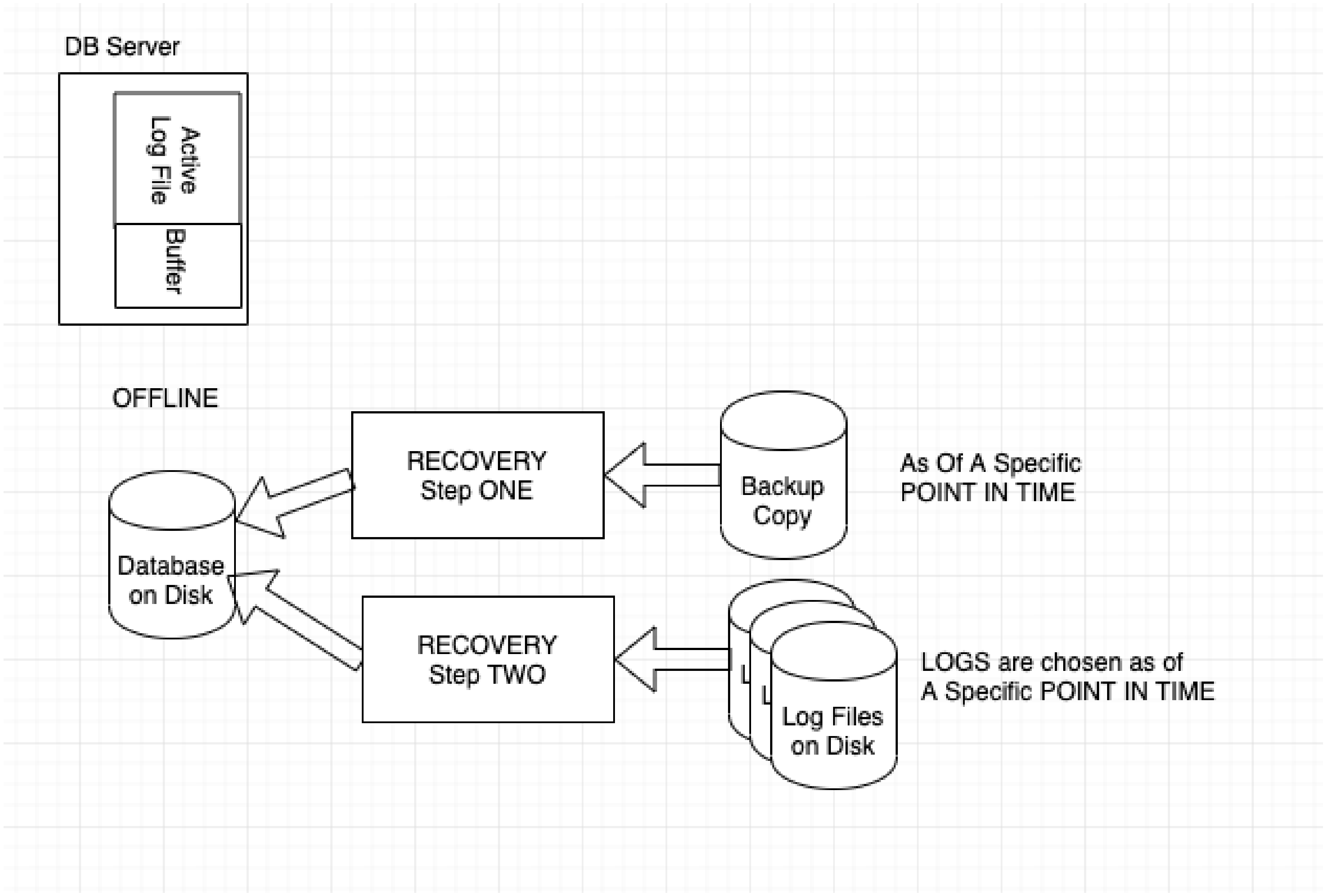
DBMS Backup

Recovering a database after a crash or corruption / DR:

1. **Identify the most recent full backup**
2. **Identify the latest transaction log file**
3. **Identify the point-in-time of the failure**
4. **Restore the most recent full backup**
5. **Apply transaction logs up to the last commit before the failure occurred**

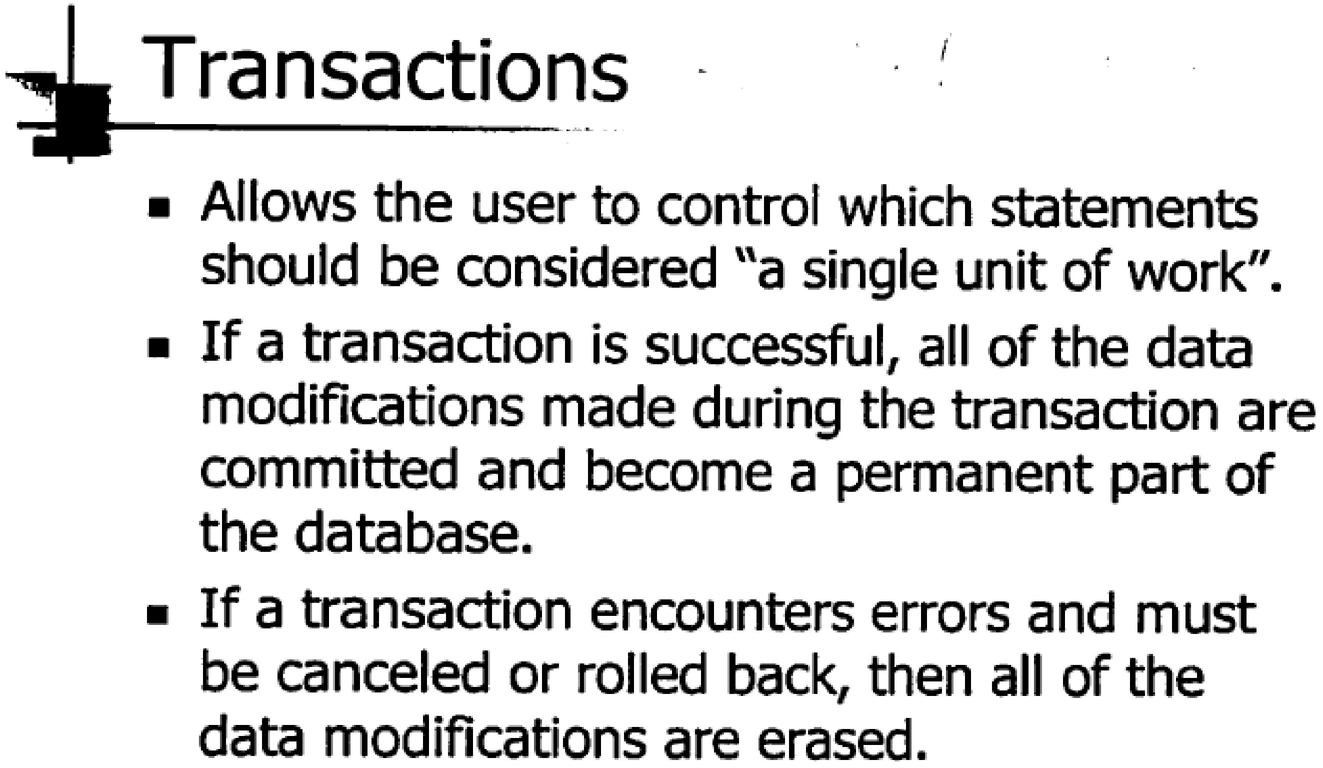
DBMS Backup

Standard Recovery:



DBMS Transaction

- **DBMS Transaction Management**
- A transaction is a logical unit of processing in a DBMS which entails one or more database access operation.
- Commit / Rollback



Transactions

- Allows the user to control which statements should be considered "a single unit of work".
- If a transaction is successful, all of the data modifications made during the transaction are committed and become a permanent part of the database.
- If a transaction encounters errors and must be canceled or rolled back, then all of the data modifications are erased.

DBMS Transaction

A transaction in a database system must maintain ACID properties

The acronym ACID

– Transaction management that is Atomic, Consistent, Isolated, and Durable.

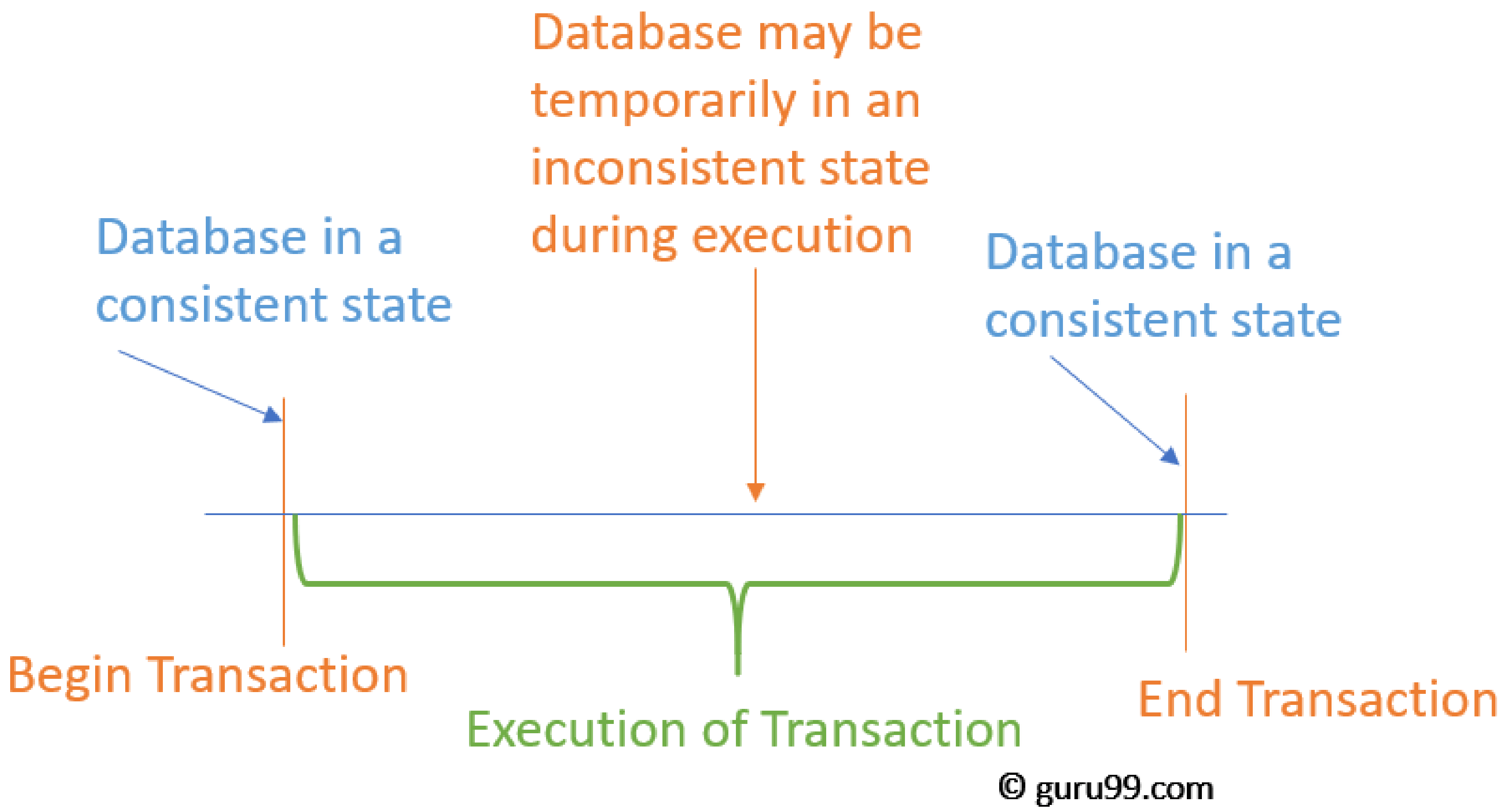
- **Atomic** means either **ALL or NONE** of the database actions within a transaction are committed.
- **Durable** means database **committed changes are permanent**.

DBMS Transaction

- **Consistency** means the DBMS software will **NOT** allow any application programs to violate **any database constraints**. Once committed, the transaction leaves the **data in a consistent state**.
- **Isolation** means the DBMS software
  - Allows application programmers are able to declare the **desired isolation level**,
  - So that they can trust that the DBMS manages locks so as to achieve that level of isolation.

DBMS Transaction

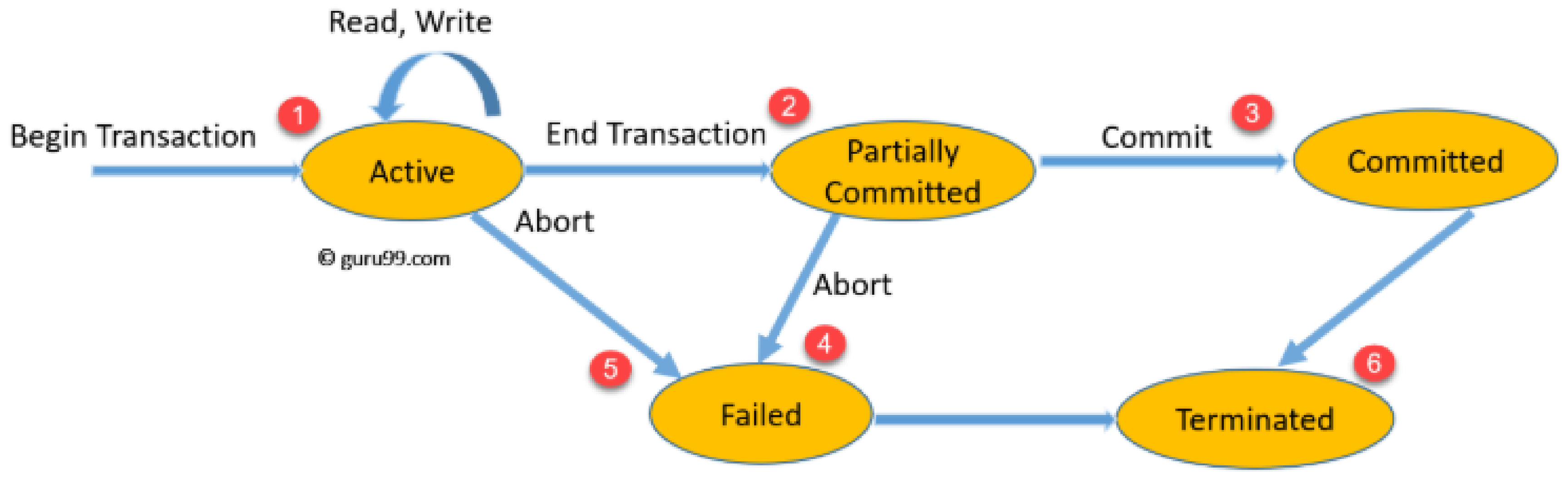
Operation between the beginning and end transaction statements are considered as a **single logical transaction**


Database may be temporarily in an inconsistent state during execution
Database in a consistent state
Database in a consistent state
Begin Transaction
Execution of Transaction
End Transaction
© guru99.com

DBMS Transaction

States of Transactions

1. Begin transaction process; 2. end of transaction; 3. changes are permanent.
4/5. transaction aborted; 6. transactions leaving the system can't be restarted.



State Transition Diagram for a Database Transaction

DBMS Transaction

Anomalies:

“Dirty Read” – a READ request is allowed to read from cache the updated but uncommitted copy of the data

“Unrepeatable Read” – One READ request following another READ request gets two different results.

Alice and Bob are using the Fandango website to book tickets for a specific movie showing. Only one ticket is left for the specific show. Alice signs on first to see that only one ticket is left, and feels like it is a bit expensive. Alice takes time to decide. Bob signs on and also finds one ticket left, and buys it instantly. Bob makes his purchase and logs off. Alice decides to buy a ticket, and finds there are no more tickets. This is a typical unrepeatable read situation.

DBMS Transaction

Anomalies:

“Phantom Read” – a READ request returns a different set of rows at different times. For example the first SELECT returns 10 rows; the next SELECT returns 11 rows (including a “phantom” row) that was inserted after the first read.

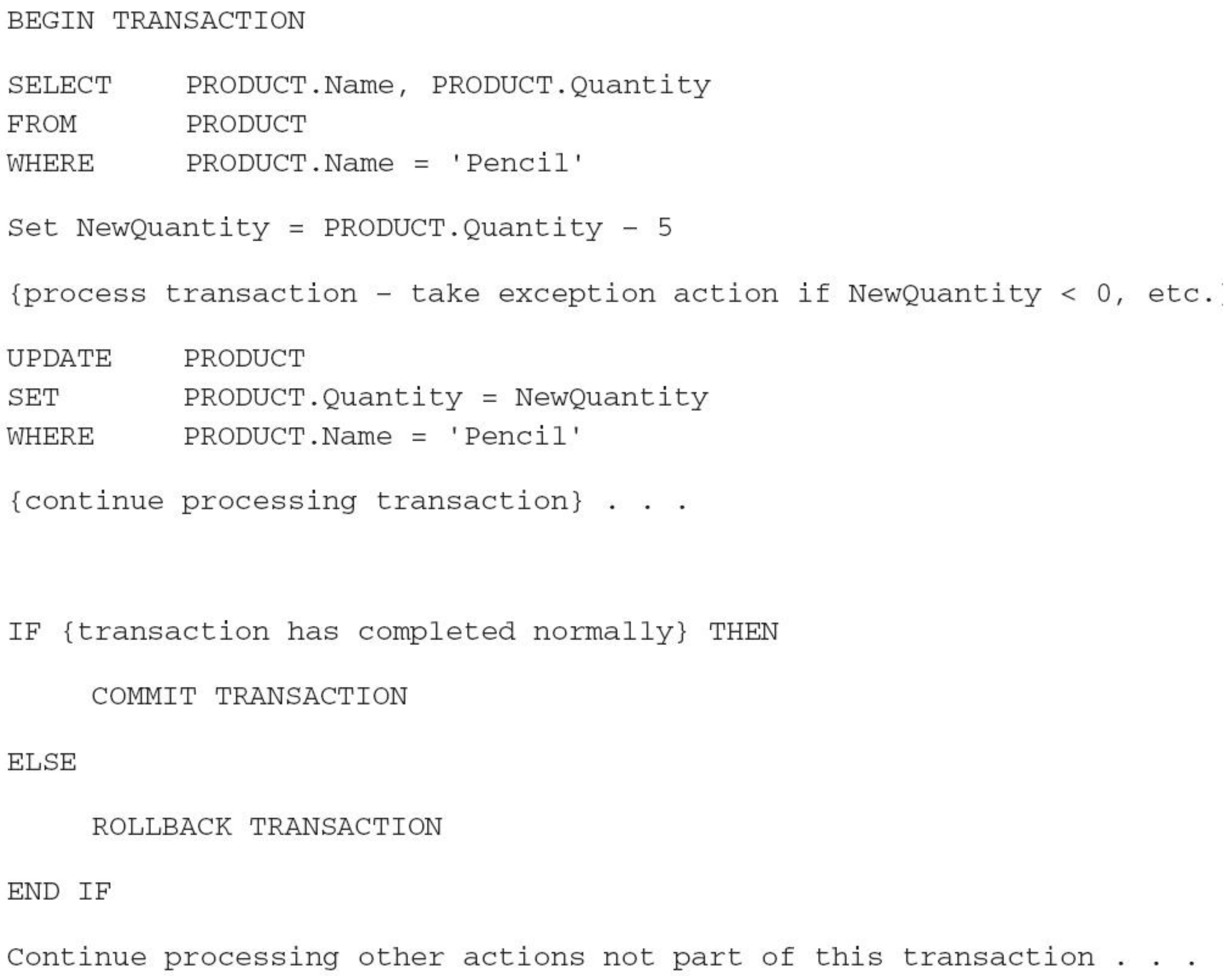
DBMS Transaction

- **The ANSI SQL-92 standard defines four transaction isolation levels:**
  - **Read uncommitted – allows reads of uncommitted changes ‘i.e. “dirty” reads, unrepeatable reads and phantom reads**
  - **Read committed – allows unrepeatable reads and phantom reads, but no dirty reads**
  - **Repeatable read – allows phantom reads, but no unrepeatable reads**
  - **Serializable – guarantees no read anomalies**

  process of a concurrent schedule, transactions are executed in a serial manner, one after the other

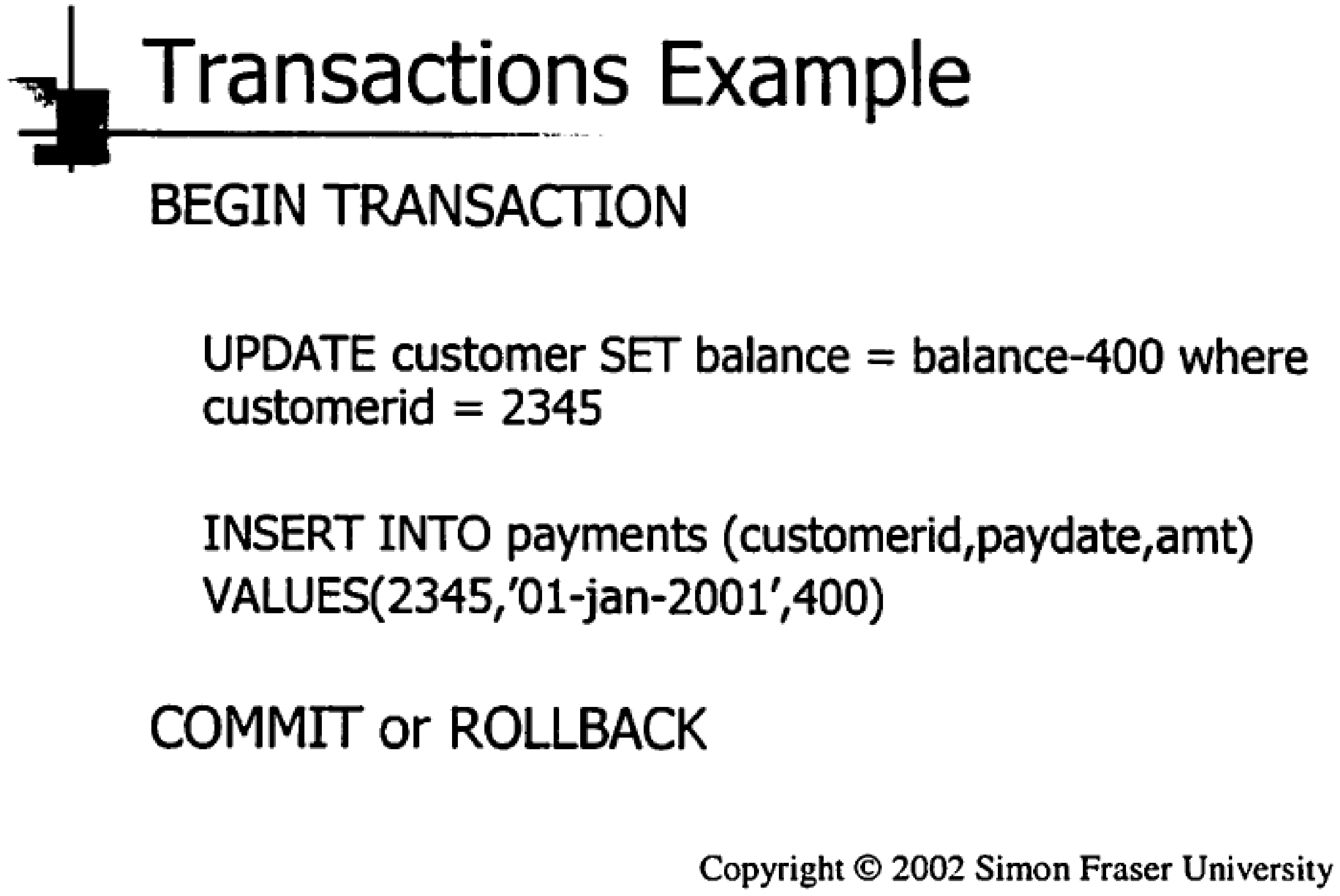
DBMS Transaction

- **Psuedo-Code**



```
BEGIN TRANSACTION

SELECT    PRODUCT.Name, PRODUCT.Quantity
FROM      PRODUCT
WHERE     PRODUCT.Name = 'Pencil'

Set NewQuantity = PRODUCT.Quantity - 5

{process transaction - take exception action if NewQuantity < 0, etc.

UPDATE    PRODUCT
SET       PRODUCT.Quantity = NewQuantity
WHERE     PRODUCT.Name = 'Pencil'

{continue processing transaction} . . .


IF {transaction has completed normally} THEN

     COMMIT TRANSACTION

ELSE

     ROLLBACK TRANSACTION

END IF

Continue processing other actions not part of this transaction . . .
```

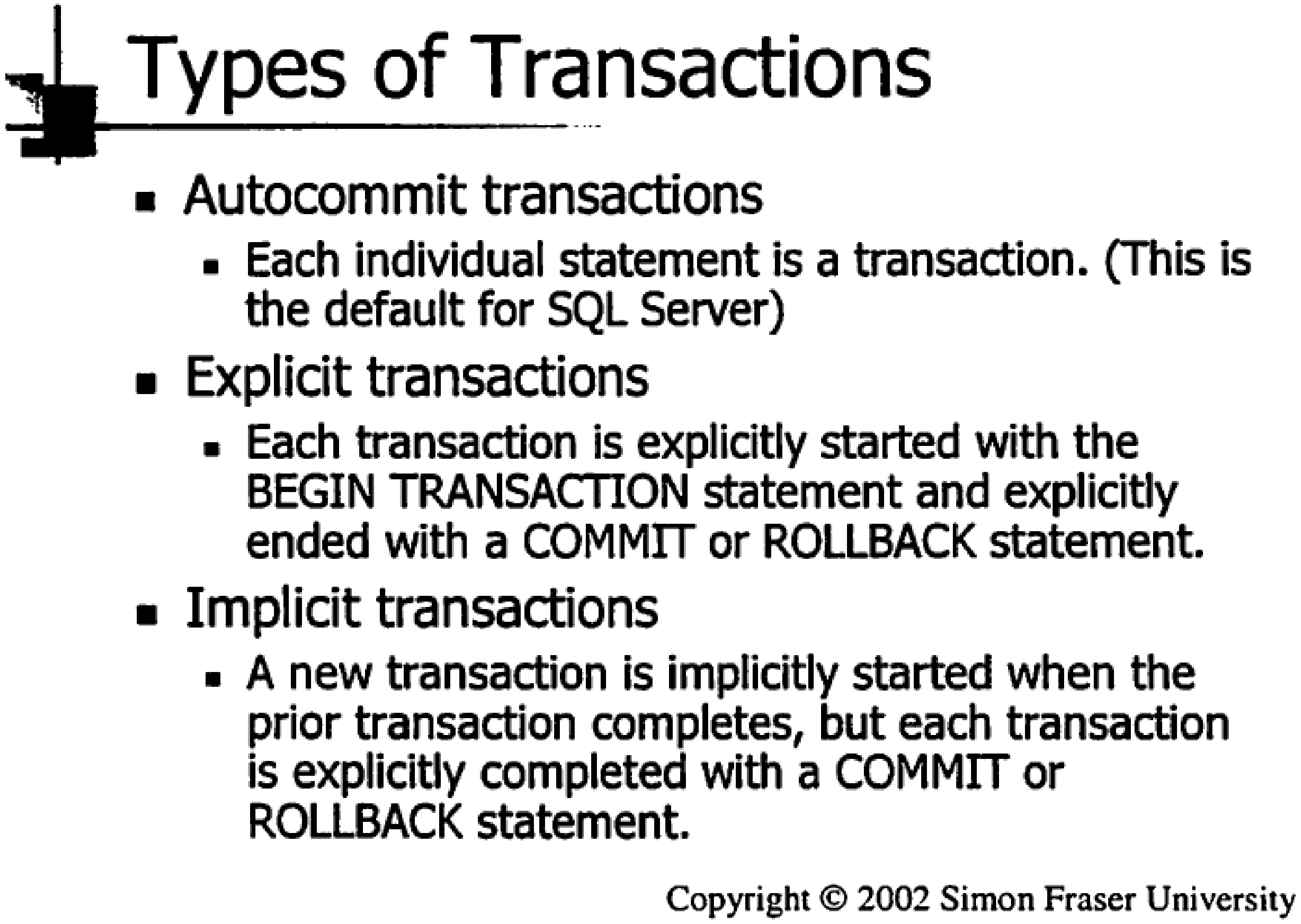
DBMS Transaction



Transactions Example

```
BEGIN TRANSACTION

    UPDATE customer SET balance = balance-400 where
    customerid = 2345

    INSERT INTO payments (customerid,paydate,amt)
    VALUES(2345,'01-jan-2001',400)

COMMIT or ROLLBACK
```

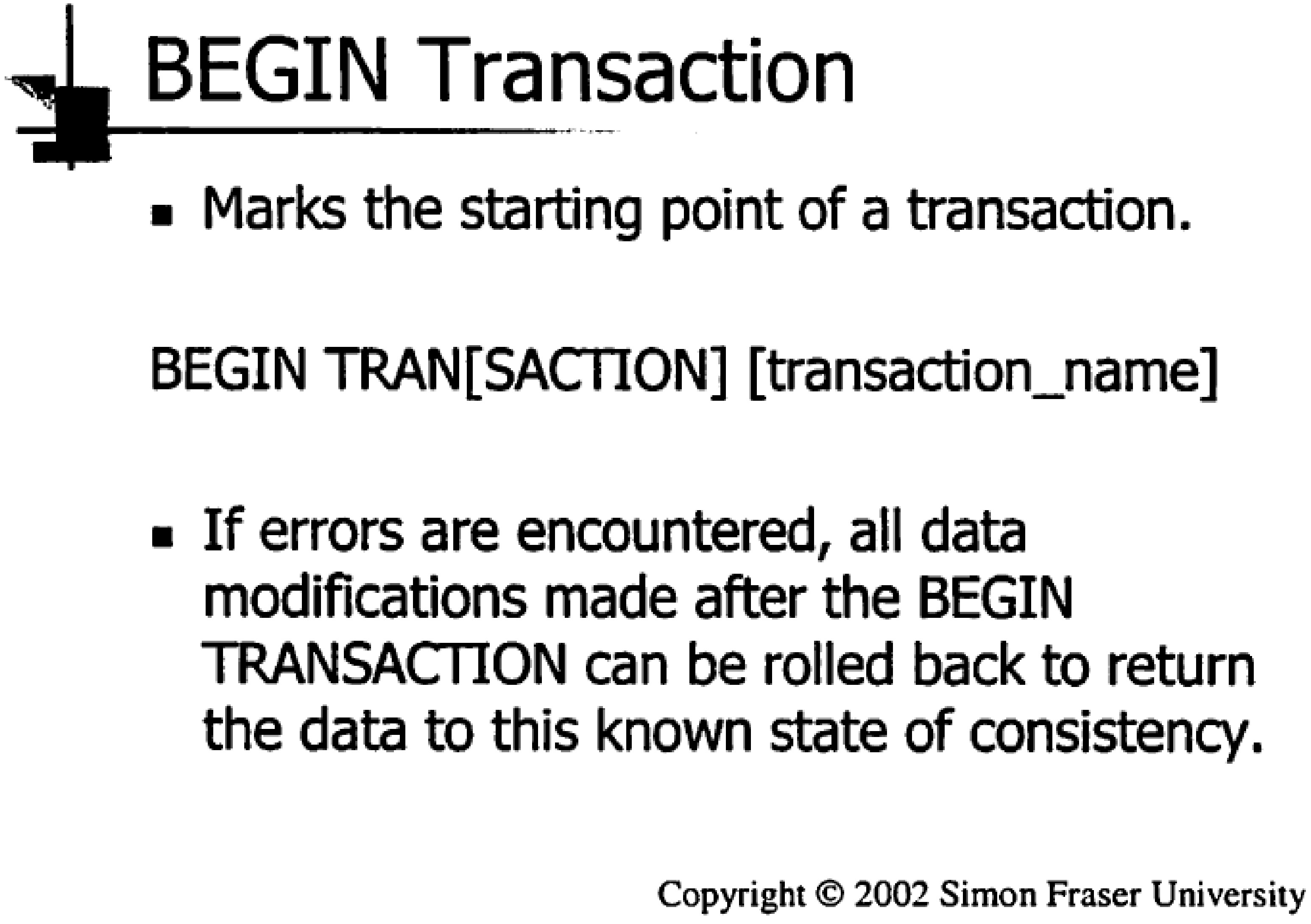
DBMS Transaction



Types of Transactions

- Autocommit transactions
  - Each individual statement is a transaction. (This is the default for SQL Server)
- Explicit transactions
  - Each transaction is explicitly started with the BEGIN TRANSACTION statement and explicitly ended with a COMMIT or ROLLBACK statement.
- Implicit transactions
  - A new transaction is implicitly started when the prior transaction completes, but each transaction is explicitly completed with a COMMIT or ROLLBACK statement.

DBMS Transaction



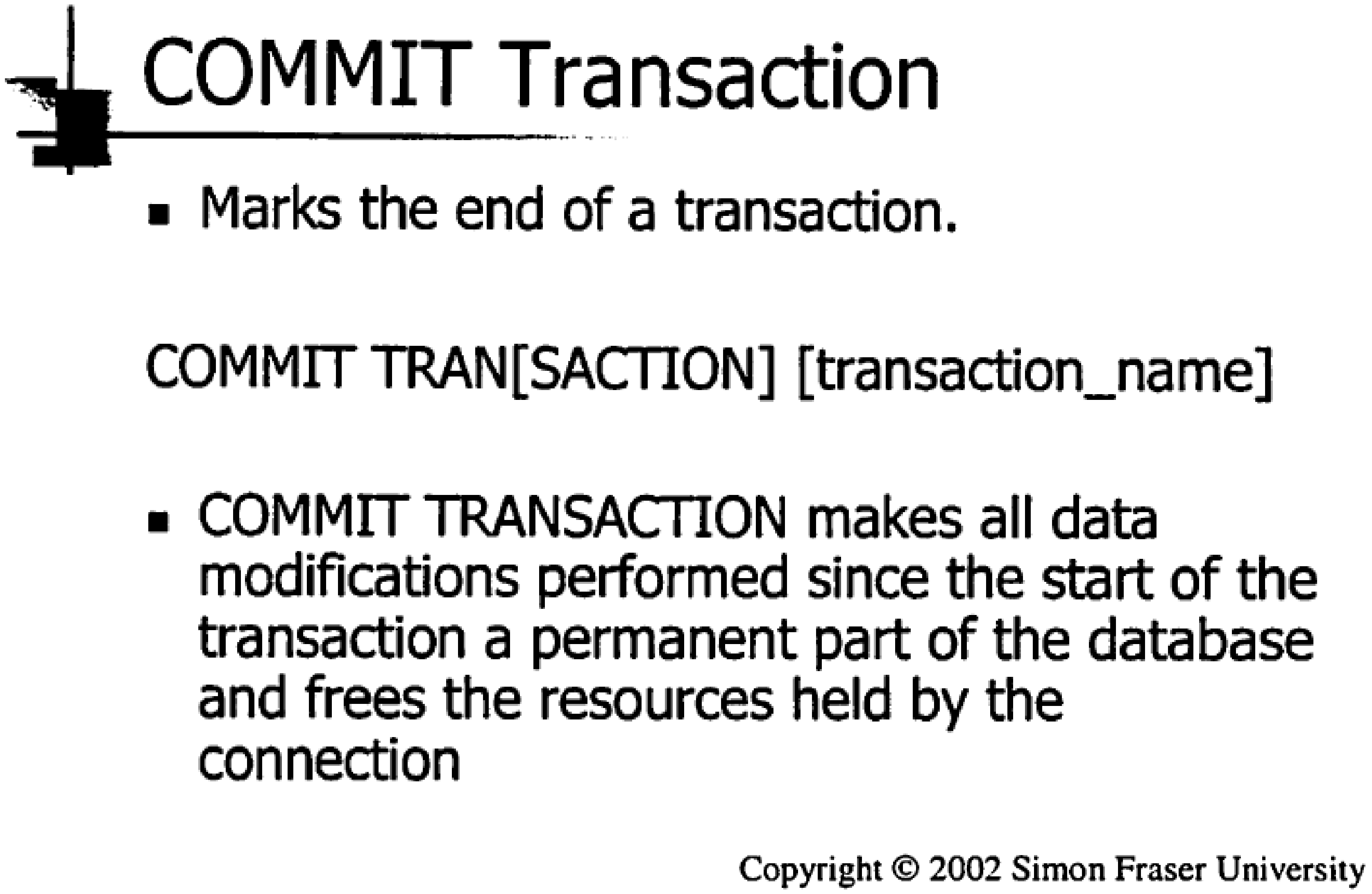
BEGIN Transaction

- Marks the starting point of a transaction.

BEGIN TRAN[SACTION] [transaction_name]

- If errors are encountered, all data modifications made after the BEGIN TRANSACTION can be rolled back to return the data to this known state of consistency.

DBMS Transaction



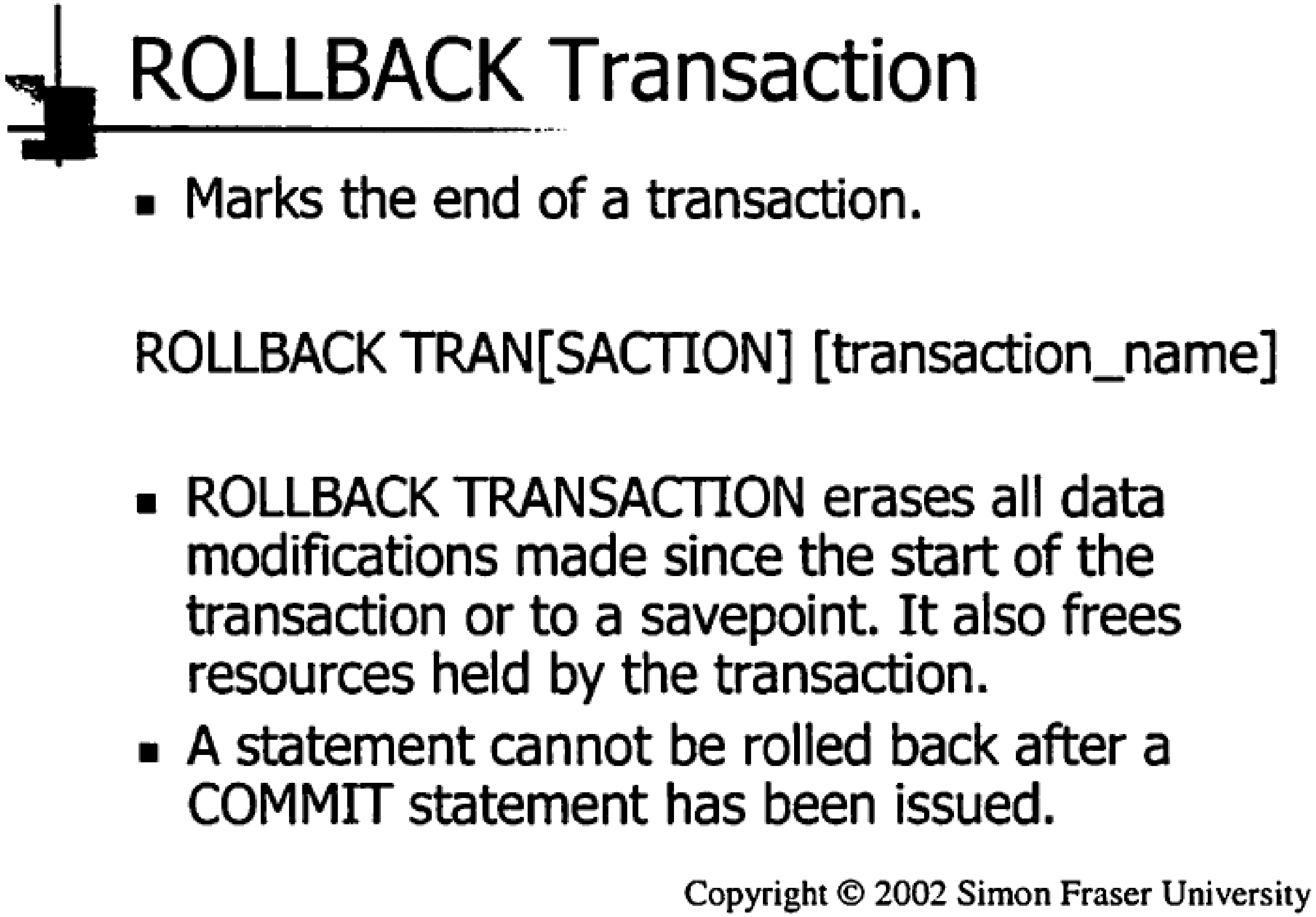
COMMIT Transaction

- Marks the end of a transaction.

COMMIT TRAN[SACTION] [transaction_name]

- COMMIT TRANSACTION makes all data modifications performed since the start of the transaction a permanent part of the database and frees the resources held by the connection

DBMS Transaction



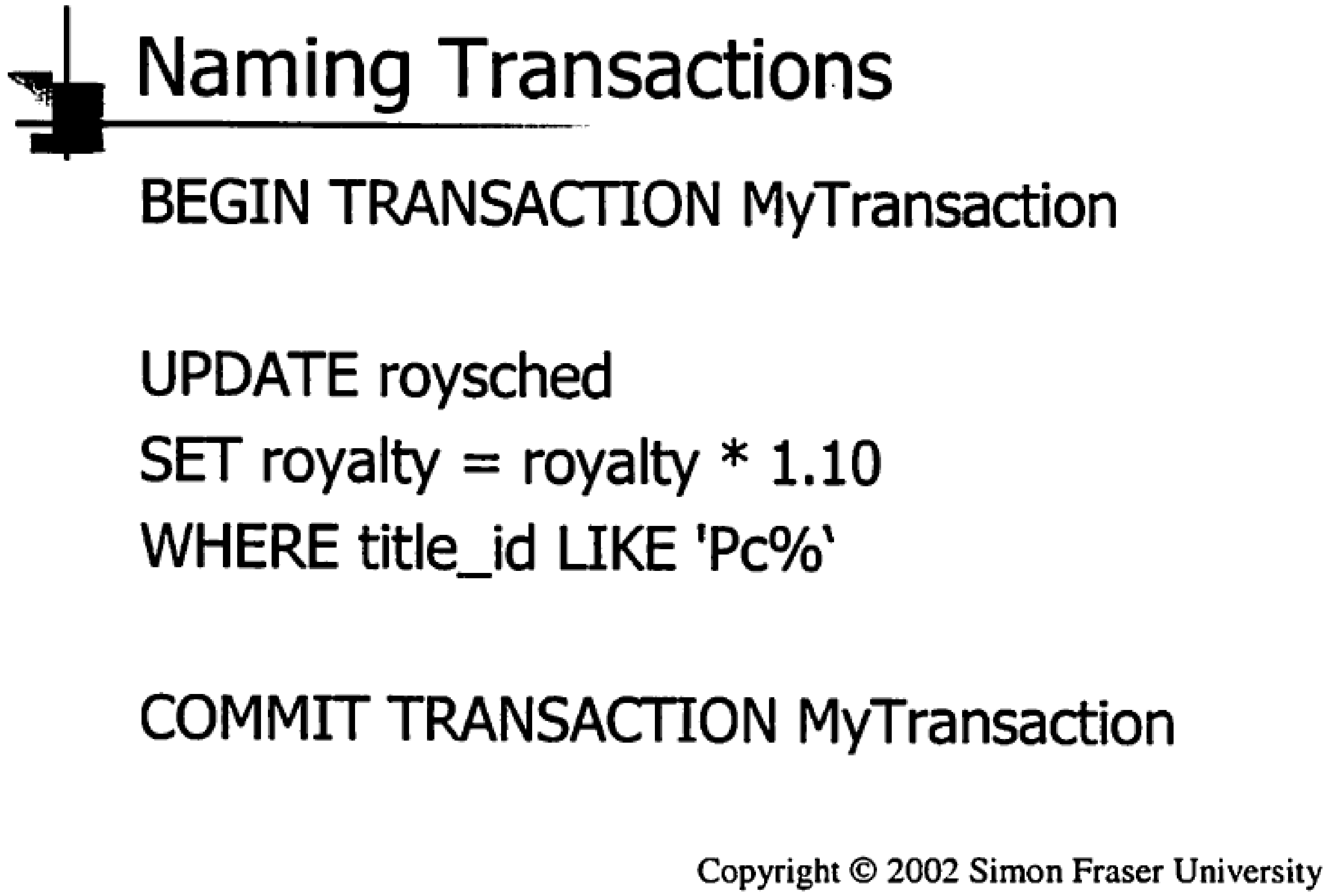
ROLLBACK Transaction

- Marks the end of a transaction.

ROLLBACK TRAN[SACTION] [transaction_name]

- ROLLBACK TRANSACTION erases all data modifications made since the start of the transaction or to a savepoint. It also frees resources held by the transaction.
- A statement cannot be rolled back after a COMMIT statement has been issued.

DBMS Transaction



Naming Transactions

```
BEGIN TRANSACTION MyTransaction

UPDATE roysched
SET royalty = royalty * 1.10
WHERE title_id LIKE 'Pc%'

COMMIT TRANSACTION MyTransaction
```