DB2 Questions for exam.

Keywords:

Alert log- is a file that provides a chronological log of database messages and errors.

Background process- Oracle Database creates background processes automatically when a database instance starts. The background processes that are present depend on the features and options that are being used in the database. The startup and shutdown of background processes are normally seen logged in the database alert.log.

Buffer- For many types of operations, Oracle Database uses the buffer cache to store data blocks read from disk. Oracle Datálásé bypasses the buffer cache for particular operations, such as sorting and parallel reads.

Cluster key - The **cluster key** is the column, or group of columns, that the clustered tables have in common. You specify the columns of the cluster key when creating the cluster. You subsequently specify the same columns when creating every table added to the cluster. Each cluster key value is stored only once each in the cluster and the cluster index, no matter how many rows of different tables contain the value

Clustered table - When tables are clustered, a single data block can contain rows from multiple tables.

Composite index - A composite index is an index on multiple columns in a table.

Control file - ***contains information about the name and location of data files, online redo log files etc.***

Data block - At the finest level of granularity, Oracle stores data in **data blocks** (also called **logical blocks**, **Oracle blocks**, or **pages**). One data block corresponds to a specific number of bytes of physical database space on disk.

Data file - *A data file is a physical file on disk that was created by Oracle Database and contains data structures such as tables and indexes.*

Database instance - A database instance is the combination of the system global area (SGA) and background processes.

Deadlock - is a situation in which two or more users are waiting for data locked by each other.

Dense index - A dense index has an index entry for every search key value.

disk block - Data blocks are mapped to disk blocks.

Dynamic sql - is SQL whose complete text is not known until run time

Extent - is a set of logically contiguous data blocks.

Function based index - Traditionally, performing a function on an indexed column in the where clause of a query guaranteed an index would not be used. Oracle 8i introduced Function-Based Indexes to counter this problem. Rather than indexing a column, you index the function on that column, storing the product of the function, not the original column data. When a query is passed to the server that could benefit from that index, the query is rewritten to allow the index to be used. The following code samples give an example of the use of Function-Based Indexes.

Heap organized table - A heap-organized table is a table with rows stored in no particular order. This is a standard Oracle table; the term "heap" is used to differentiate it from an index-organized table or external table.If a row is moved within a heap-organized table, the row's ROWID will also change.

Hint - Hints let you make decisions usually made by the optimizer. As an application designer, you might know information about your data that the optimizer does not know. Hints provide a mechanism to instruct the optimizer to choose a certain query execution plan based on the specific criteria.

Index organized table - Index Organized Tables (IOT) have their primary key data and non-key column data stored within the same B*Tree structure. Effectively, the data is stored within the primary key index.

Multi-level index -

Nonvolatile device -

Online redo log - *online redo log is a set of files containing records of changes made to data. Online redo log is the most crucial structure for recovery.*

Partition -

Partitioned index - is an index that has been divided into smaller and more manageable pieces.

Partitioned table - If you create a partitined table, then no table segment is allocated for this table.

Partitioning - makes it possible to decompose very large tables and indexes into smaller and more manageable pieces

Reverse key index - A reverse key index stores the index entries with their bytes reversed. The ROWIDs are stored in the same format as a regular index. When you insert rows in a column where the database populates one of the columns using an increasing sequence, each new entry will be inserted into the same index block. When each new key value in an index is greater than the previous value, it is said to be a monotonically increasing value**.**

Rotational latency - ***is the time while the first sector of the block moves under the head.***

Secondary index - In case of a secondary index, data records are not sorted by the search key.

Secondary storage -

Seek time -

Segment - Segments exist within a tablespace. Segments are made up of a collection of extents.

Sequence - A ... is a schema object from which multiple users can generate unique integers.

Sequential file- A ... file means that we sort the tuples of a relation by their primary

key and store them in data blocks in this order.**

Sparse index- A sparse (or nondense) index, on the other hand, has index entries for only some of the search values (typically one entry per data file block)

Subpartition - In composite partitioning each partition is further divided into subpartition

Synonym- ***is useful for hiding the identity and location of an underlying schema object.***

System global area - The System Global Area (SGA) is a group of shared memory structures, known as SGA components, that contain data and control information for one Oracle Database instance. The SGA is shared by all server and background processes. Examples of data stored in the SGA include cached data blocks and shared SQL areas.

Table cluster - is a group of tables that share common columns and store related data in the same blocks.

Tablespace - The database data files are logically grouped together in a tablespace.

Temp file - A tempfile is a file that is part of an Oracle database. Tempfiles are used with TEMPORARY TABLESPACES and are used for storing temporary data like sort spill-over or data for global temporary tables.

Temporary tablespace - tablespace is a tablespace that can only contain transient data that persists only for the duration of a session.

Tertiary storage - is characterized by higher read/write times than secondary storage.

Transfer time -

View - In Oracle, view is a virtual table that does not physically exist. It is stored in Oracle data dictionary and do not store any data. It can be executed when called. **And requires no storage other than the storage of its query in the data dictionary.**

Volatile device - A volatile device "forgets" what is stored in it when the power goes off.

Questions:

What are the most important files (file types) in an Oracle database? (01_Oracle_storage.pptx 1.)

A data file is a physical file on disk that was created by Oracle Database and contains data structures such as tables and indexes.

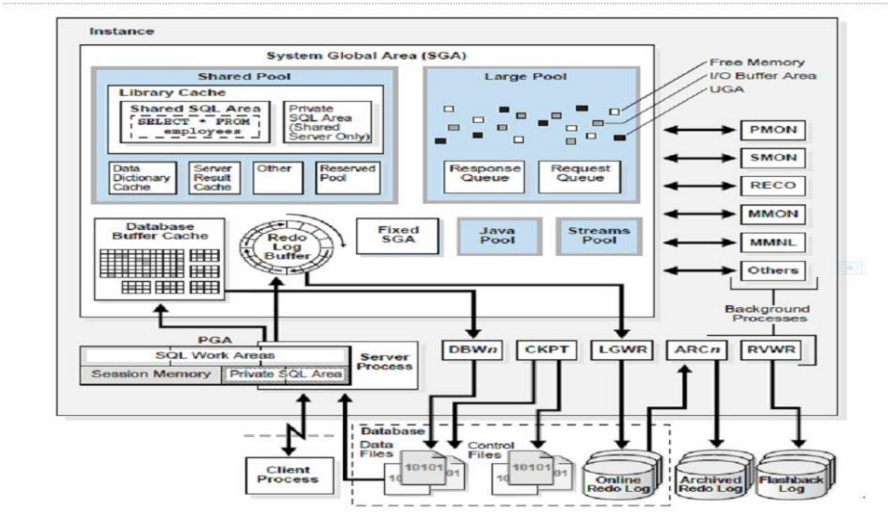
A control file contains information such as the following: the database name, information about data files, online redo log files, tablespace information, etc.

The online redo log is a set of files containing records of changes made to data. Online redo log is the most crucial structure for recovery.

Alert log is a file that provides a chronological log of database messages and errors.

Give the most important memory structures of an Oracle instance. (01_Oracle_architecture.pptx 5.)

- *Software Code Areas*
- *System Global Area (SGA):*
  - *the database buffer cache*
  - *the redo log buffer*
  - *the shared pool*
- *Program Global Areas (PGA):*
  - *the stack areas*
  - *the data areas*
- *Sort Areas*



Give the most important processes of an Oracle instance. (01_Oracle_architecture.pptx 5.)

- *Database Writer Process (DBW**n**)*
- *Log Writer Process (LGWR)*
- *Checkpoint Process (CKPT)*
- *System Monitor Process (SMON)*
- *Process Monitor Process (PMON)*

- *Recoverer Process (RECO)*
- *Job Queue Processes*
- *Archiver Processes (ARC**n**)*
- *Queue Monitor Processes (QMN**n**)*

List 5 data dictionary views in an Oracle database.

USER,ALL,DBA,V$,_PRIVS

The data dictionary views, also known as catalog views, let you monitor the state of the database in real time:

- *The USER, ALL, and DBA, views show information about schema objects that are accessible to you, at different levels of privilege.*
- *The V$ views show performance-related information.*
- *The _PRIVS views show privilege information for different combinations of users, roles, and objects.*

List 10 different schema objects in an Oracle database.

- *Introduction to Schema Objects*
- *Tables*
- *Views*
- *Materialized Views*
- *Dimensions*
- *The Sequence Generator*
- *Synonyms*
- *Indexes*
- *Index-Organized Tables*
- *Application Domain Indexes*
- *Clusters*
- *Hash Clusters*

List 5 different objects in an Oracle database which are not in a user's schema.

dba_objects?

???

What is a sequence in an Oracle database? Give SQL examples for the creation and usage. (01_Oracle_architecture.pptx 15-17.)

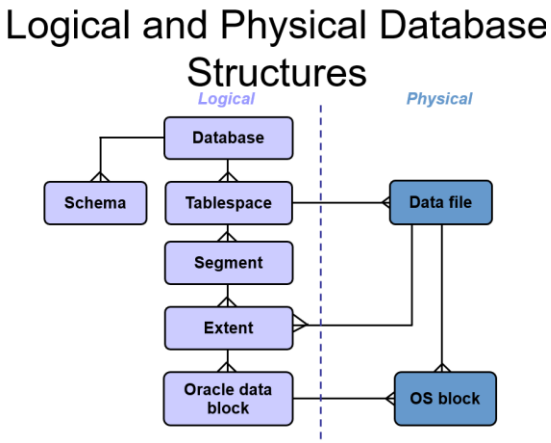
- A sequence is a mechanism for automatically generating integers that follow a pattern.
  - A sequence has a name, which is how it is referenced when the next value is requested.
  - A sequence is not associated with any particular table or column.
  - The progression can be ascending or descending.
  - The interval between numbers can be of any size.
  - A sequence can cycle when a limit is reached.

Example :

```
CREATE SEQUENCE seq1
MINVALUE 1 MAXVALUE 100 INCREMENT BY 5
START WITH 50 CYCLE; --creation
SELECT * FROM DBA_SEQUENCES
WHERE sequence_name='SEQ1'; --usage
```

Give the data storage concepts (segment, extent etc.) in an Oracle database, and draw the relationships among them. (01_Oracle_storage.pptx 8.) :

- Segments exist within a tablespace.
- Segments are made up of a collection of extents.
- Extents are a collection of data blocks.
- Data blocks are mapped to disk blocks.



Describe RAID level 0, 1, 2, 3, 4, 5, 6 technology. (02_RAID.docx)

RAID 0 (block level striping, no parity bits, no mirroring)

RAID 1 (mirroring, no parity bits, no striping)

RAID 2 (bit level striping, error correcting parity bits)

RAID 3 (byte level striping, dedicated parity disk)

RAID 4 (block level striping, dedicated parity disk)

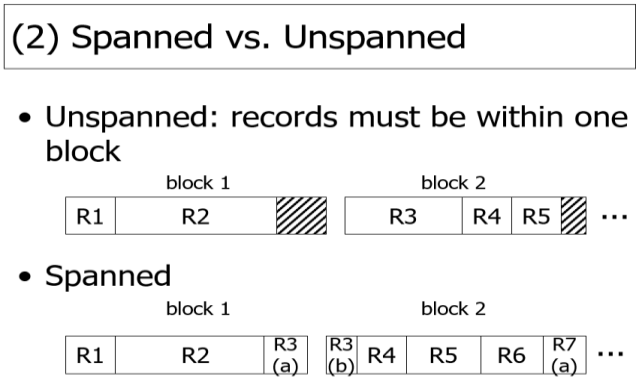
RAID 5 (block level striping, distributed parity)

RAID 6 (block level striping, double parity bits, distributed parity disks)

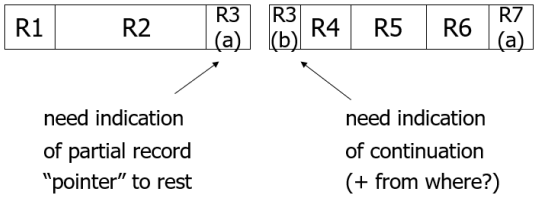
What does it mean: spanned vs unspanned record? Draw example data blocks for both. (02_UW_file_structure 27-29.)

- Unspanned is much simpler, but may waste space…
- Spanned essential if

  record size > block size



With spanned records:



Give 3 sequencing options for records. (02_UW_file_structure 31-33.)

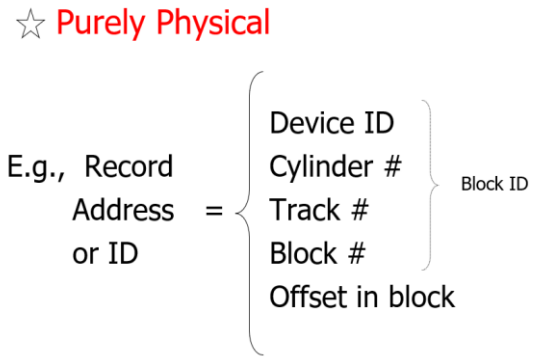
Next record physically contiguous

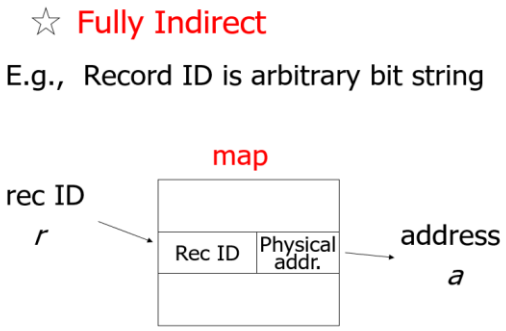
Linked

Overflow area (records in sequence)

What is the difference between a purely physical and a fully indirect record reference? (02_UW_file_structure 35-37.)

Flexibility to move records(for deletions, insertion) ⇔ cost of indirection(manage the map)





Describe the difference between row store and column store. Give example records for both. (02_UW_file_structure 55-58.)

- So far, we assumed that fields of a record are stored contiguously (row store)...
- Another option is to store like fields together (column store)

Row Store

- Example: Order consists of
  - id, cust, prod, store, price, date, qty

id1	cust1	prod1	store1	price1	date1	qty1
id2	cust2	prod2	store2	price2	date2	qty2
id3	cust3	prod3	store3	price3	date3	qty3

Column Store

- Example: Order consists of
  - id, cust, prod, store, price, date, qty

id1	cust1
id2	cust2
id3	cust3
id4	cust4
...	...

id1	prod1
id2	prod2
id3	prod3
id4	prod4
...	...

id1	price1	qty1
id2	price2	qty2
id3	price3	qty3
id4	price4	qty4
...	...	...

ids may or may not be stored explicitly

What is the difference between a sparse index and a dense index? (03_UW_indexing 5.)

- Indexes can also be characterized as dense or sparse
  - A dense index has an index entry for every search key value (usually every record) in the data file.
  - A sparse (or nondense) index, on the other hand, has index entries for only some of the search values (typically one entry per data file block)

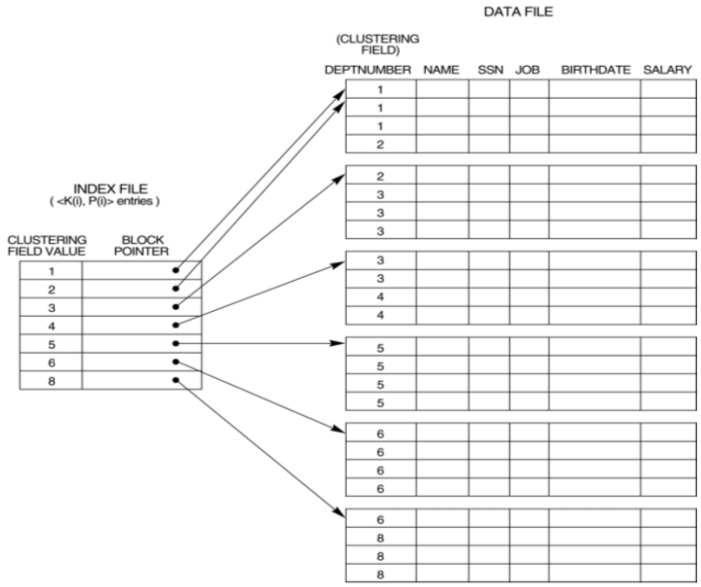
What is the difference between a primary index and secondary index? (03_UW_indexing 12.)

- Primary index (on ordering field)
- Secondary index (on non-ordering field)

What is a clustering index? (03_UW_indexing.ppt 49.)

- Clustering Index
  - Defined on an ordered data file
  - The data file is ordered on a *non-key field*.
  - Includes one index entry *for each distinct value* of the field; the index entry points to the first data block that contains records with that field value.
  - It is an example of *non-dense* index where Insertion and Deletion is relatively straightforward with a clustering index.

- Clustering Index



Insert the following keys into a given B+ tree ...

Question from midterm:

Insert the following keys (in the given order) into the B+ tree below: 52, 19, 90

Let's suppose that a node (block) can contain 3 keys and 4 pointers.

Redraw the tree after each split.

```
                72
          43|56          79
 15|39|40  43|45|49  56|70  72|75  79|80|83
```

Solution:

```
                72
          43|49|56          79
 15|39|40  43|45  49|52  56|70  72|75  79|80|83


                49|72
```

39|43 56 79

15|19 39|40 43|45 49|52 56|70 72|75 79|80|83

49|72

39|43 56 79|83

15|19 39|40 43|45 49|52 56|70 72|75 79|80 83|90

What is the difference between a B-tree and a B+ tree? (03_UW_indexing 88.)

Variation on B+tree: B-tree (no +)

- Idea:
  - Avoid duplicate keys
  - Have record pointers in non-leaf nodes

What is a bitmap index? What are there in the leaf nodes? (04_Bitmap_indexes)

These are several types of index structures available to you, depending on the need:

- A B+-tree index is in the form of a balanced tree and is the default index type.
- A bitmap index has a bitmap for each distinct value indexed, and each bit position represents a row that may or may not contain the indexed value. This is best for low-cardinality columns.

Structure of a bitmap index

A bitmap index is also organized as a B-tree, but the leaf node stores a bitmap for each key value instead of a list of ROWIDs. Each bit in the bitmap corresponds to a possible ROWID, and if the bit is set, it means that the row with the corresponding ROWID contains the key value.

As shown in the diagram, the leaf node of a bitmap index contains the following:

An entry header that contains the number of columns and lock info

Key values consisting of length and value pairs for each key column

Start ROWID

End ROWID

A bitmap segment consisting of a string of bits. (The bit is set when the corresponding row contains the key value and is unset when the row does not contain the key value. The Oracle server uses a patented compression technique to store bitmap segments.)

Compress the following bitvector with run-length encoding ...

a) Compress the following bit vector with run-length encoding:

10000000000000001100000000001

a) compressed -> 00 11101111 00 11101010

- ---- - ----

0 15 0 10

Decompress the following compressed bitvector ...

b) Decompress the following run-length encoded bit vector:

111011011010110101

b) 111011011010110101

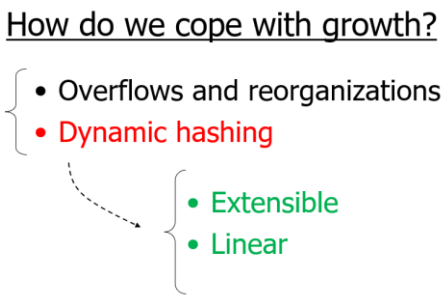
---- -- ---

13 2 5

decompressed -> 00000000000001001000001

What is dynamic hashing? (04_UW_hashing 18.)

- The dynamic hashing method is used to overcome the problems of static hashing like bucket overflow.
- In this method, data buckets grow or shrink as the records increases or decreases. This method is also known as Extendable hashing method.
- This method makes hashing dynamic, i.e., it allows insertion or deletion without resulting in poor performance.



Build a linear hash structure from the following key values ...

???

Build an extensible hash structure from the following key values ...

see example 04_UW_hashing.pptx slides 22-30

What is the most important cost factor in query execution? (05_optimization 9.)

- What needs to be considered:
  - Disk I/Os
    - sequential (reading neighboring pages is faster)
    - random (reading pages in the order of requests)
  - CPU time
  - Network communication
- What are we going to consider:
  - Disk I/Os
    - page (data block) reads/writes
  - Ignoring cost of writing final output

Give the meaning of the following notations that we use in cost estimation: T(R), B(R), bf(R), V(R,A), SC(R,A). (05_optimization 10.)

- Costs:
  - N_R: number of records in R (other notation: T_R or T(R) -> tuple)
  - L_R: size of record in R (length of record -> L(R))
  - bf_R: blocking factor (other notation: F_R or bf(R))
    - number of records in a page (datablock)
  - B_R: number of pages to store relation R (-> B(R))
  - V(R, A): number of distinct values of attribute A in R
    other notation: $I_A(R)$ (Image size)
  - SC(R,A): selection cardinality of A in R (number of matching rec.)
    - A key: SC(R, A)=1
    - A nonkey: SC(R, A)= T_R / V(R,A) (uniform distribution assumption)
  - HT_i: number of levels in index I (-> height of tree)
  - rounding up fractions and logarithms

What is the average cost of a selection operation (σA=xR) if we use a clustered B+ tree index? in case of single record/multiple record (05_optimization 12.)

- Primary/Clustered Index (B+ tree)
  - average cost:

- single record: $HT_i + 1$
- multiple records: $HT_i + ceil(SC(R,A)/bf_R)$

What is the average cost of a selection operation (σA=xR) if we use a secondary B+ tree index? in case of key field/nonkey filed (05_optimization 13.)

- Secondary Index (B+ tree)
  - average cost:
    - key field: $HT_i + 1$
    - nonkey field
      - worst case $HT_i + SC(A,R)$
      - linear search more desirable if many matching records !!!

Describe the external Sort-Merge algorithm. What is the cost of it? (05_optimization 21.)

- Sort stage: create sorted *runs*

i=0;

repeat

read *M* pages of relation R into memory (M: size of Memory)

sort the *M* pages

write them into file R_i

increment i

until no more pages

N = i // number of runs

- Merge stage: merge sorted *runs*

//assuming N < M (N <= M-1 we need 1 output buffer)

allocate a page for each run file R_i // N pages allocated

read a page P_i of each R_i

repeat

choose first record (in sort order) among N pages, say from page P_j

write record to output and delete from page P_j

if page is empty read next page P_j' from R_j

until all pages are empty

- Merge stage: merge sorted *runs*
- What if N >= M ?
  - perform multiple *passes*
  - each *pass* merges M-1 runs until relation is processed
  - in next pass number of runs is reduced
  - final *pass* generated sorted output

Sort-Merge cost

- B_R the number of pages of R
- Sort stage: $2 * B_R$
  - read/write relation
- Merge stage:
  - initially $\left\lceil \frac{B_R}{M} \right\rceil$ runs to be merged
  - each *pass* M-1 runs sorted
  - thus, total number of passes: $\left\lceil \log_{M-1}\left(\frac{B_R}{M}\right) \right\rceil$
  - at each pass $2 * B_R$ pages are read/written
    - read/write relation ($B_R + B_R$)
    - apart from final write (B_R)
- Total cost:
  - $2 * B_R + 2 * B_R * \left\lceil \log_{M-1}\left(\frac{B_R}{M}\right) \right\rceil - B_R$ eg. $B_R = 1000000$, M=100

What is the cost of a Nested Loop join algorithm? (best case/worst case) (05_optimization 25.)

- R ⋈ S

  for each tuple t_R of R

  for each t_S of S

  if (t_R t_S match) output $t_R.t_S$

  end

  end
- Works for any join condition
- S inner relation
- R outer relation

Costs:

- best case when smaller relation fits in memory
    - use it as inner relation
    - B_R+B_S
- worst case when memory holds one page of each relation
    - S scanned for each tuple in R
    - $T_R * B_s + B_R$

What is the cost of a Block Nested Loop join algorithm? (best case/worst case) (05_optimization 27.)

- Costs:
    - best case when smaller relation fits in memory
        - use it as inner relation
        - B_R+B_S
    - worst case when memory holds one page of each relation
        - S scanned for each page in R
        - $B_R * B_s + B_R$

What is the cost of the improved Block Nested Loop join algorithm? (05_optimization 29.)

- Costs:
    - best case when smaller relation fits in memory
        - use it as inner relation
        - B_R+B_S
    - general case
        - S scanned for each M-1 size chunk in R
        - $(B_R / (M-1)) * B_s + B_R$

Describe the Indexed Nested Loop join algorithm? What is the cost of it? (05_optimization 30.)

- R $\triangleright\!\!\bowtie\!\!\triangleleft$ S
- Index on inner relation (S)
- for each tuple in outer relation (R) *probe* index of inner relation

- Costs:
  - $B_R + T_R * c$
    - c the cost of index-based selection of inner relation

$c \approx T(S)/V(S,A)$

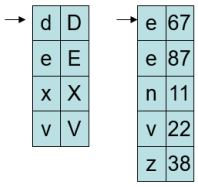
(if A is the join column and index is kept in memory)

  - relation with fewer records as outer relation

Describe the Sort-merge join algorithm. What is the cost of it? (05_optimization 31.)

Sort-merge join

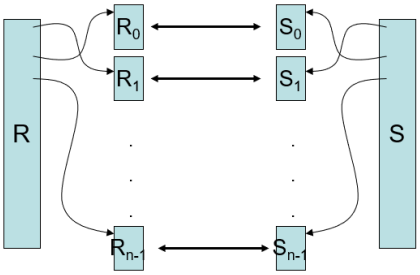
- R ⋈S
- Relations sorted on the join attribute
- Merge sorted relations
  - pointers to first record in each relation
  - read in a group of records of S with the same values in the join attribute
  - read records of R and process
- Relations in sorted order to be read once
- Cost:
  - cost of sorting + $B_S + B_R$



Describe the Hash-join algorithm? What is the cost of it? (05_optimization 32.)

Hash join

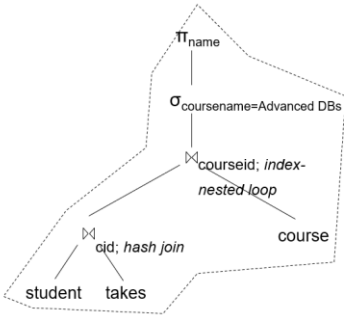
- R ⋈ S
- use h_1 on joining attribute to map records to partitions that fit in memory
  - records of R are partitioned into $R_0 \ldots R_{n-1}$
  - records of S are partitioned into $S_0 \ldots S_{n-1}$
- join records in corresponding partitions
  - using a hash-based indexed block nested loop join
- Cost: $2*(B_R+B_S) + (B_R+B_S)$



What is materialization and pipelining? (05_optimization 33.)

Evaluation

- evaluate multiple operations in a plan
- materialization
- pipelining

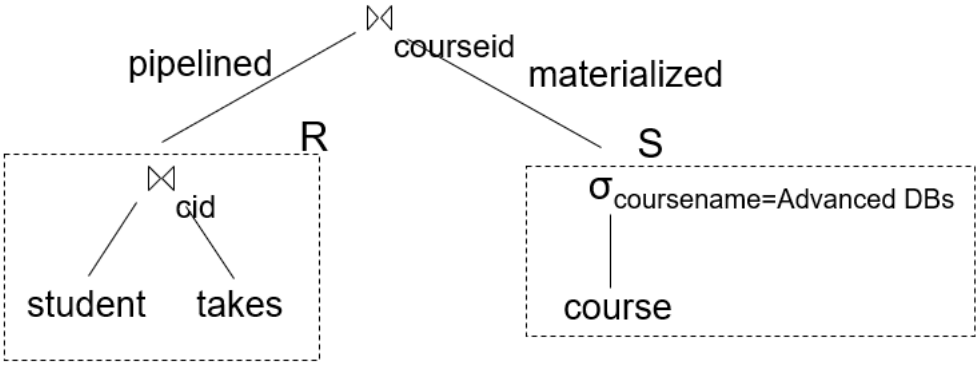


Materialization

- create and read temporary relations
- create implies writing to disk
  - more page writes

Pipeliling

- creating a pipeline of operations
- reduces number of read-write operations
- implementations
  - demand-driven - data *pull*
  - producer-driven - data *push*
- can pipelining always be used?
- any algorithm?
- cost of R S
  - materialization and hash join: $B_R + 3(B_R+B_S)$
  - pipelining and indexed nested loop join: $T_R * HT_i$



Give some basic relational algebra expression equivalence rules (05_optimization 39.) conjunctive selection decomposition (05_optimization 39.) distribution of selection over join (05_optimization 39.) distribution of projection over join (05_optimization 39.) associativity of joins, products, union (05_optimization 39.)

Expression Equivalence

- conjunctive selection decomposition
  - $\sigma_{\theta_1 \wedge \theta_2}(R) = \sigma_{\theta_1}(\sigma_{\theta_2}(R))$
- commutativity of selection
  - $\sigma_{\theta_1}(\sigma_{\theta_2}(R)) = \sigma_{\theta_2}(\sigma_{\theta_1}(R))$
- combining selection with join and product
  - $\sigma_{\theta 1}(R \times S) = \ldots \quad R \bowtie_{\theta 1} S = \ldots$
- commutativity of joins
  - $R \bowtie_{\theta 1} S = S \bowtie_{\theta 1} R$
- distribution of selection over join
  - $\sigma_{\theta 1 \wedge \theta 2}(R \bowtie S) = \sigma_{\theta 1}(R) \bowtie \sigma_{\theta 2}(S)$
- distribution of projection over join
  - $\pi_{A1,A2}(R \bowtie S) = \pi_{A1}(R) \bowtie \pi_{A2}(S)$
- associativity of joins: $R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T$

Give the meaning of the following index options: (05_special_storage 8.) composite index function-based index compressed index

?

What is a partitioned table? (05_special_storage 11.)

?

List the partitioning types an Oracle database supports (05_special_storage 12-13.)

?

Give the properties of an Index-Organized Table (05_special_storage 16.)

?

Give a diagram about two clustered tables (05_special_storage 18.)

?

Give estimation for the number of blocks of a product operation: B(R x S) (06_UW_query_proc 47.)

Size estimates for W = R x S

$$T(W) = T(R) \times T(S)$$

$$L(W) = L(R) + L(S)$$
$$bf(W) = b/(L(R)+L(S))$$

$$B(W) = T(R)*T(S)/bf(W) =$$
$$= T(R)*T(S)*L(S)/b + T(S)*T(R)*L(R)/b =$$
$$= T(R)*T(S)/bf(S) + T(S)*T(R)/bf(R) =$$
$$= T(R)*B(S) + T(S)*B(R)$$

What is Selection Cardinality? (06_UW_query_proc 50.)

Selection cardinality

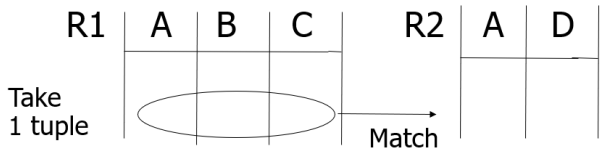
SC(R,A) = average # records that satisfy equality condition on R.A

SC(R,A) = T(R) / V(R,A)

Give estimation for the number of rows of a join operation: T(R ⋈S) (06_UW_query_proc 54.)

? not sure ?

Computing T(W) when V(R1,A) ≤ V(R2,A)



1 tuple matches with $\frac{T(R2)}{V(R2,A)}$ tuples...

so $T(W) = \frac{T(R2)}{V(R2, A)} \times T(R1)$

Give estimation for the number of rows of σA<=xR (06_UW_query_proc 59.)

Give estimation for the number of rows of σΘ1∧Θ2∧ … ∧ΘnR (06_UW_query_proc 59.)

Give estimation for the number of rows of σΘ1∨Θ2∨ … ∨ΘnR (06_UW_query_proc 59.)

Size Estimation Summary (1/2)

$\sigma_{A=v}(R)$ SC(R,A) (--> SC(R,A) = T(R) / V(R,A))

$\sigma_{A\le v}(R)$ $T(R)*\frac{v-\min(A,R)}{\max(A,R)-\min(A,R)}$

$\sigma_{\theta_1\wedge\theta_2\wedge\ldots\wedge\theta_n}(R)$ multiplying probabilities

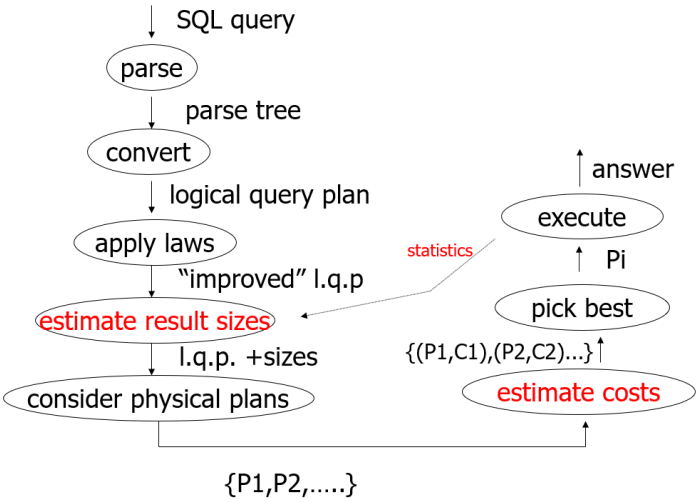
$$T(R)*[(sc_1/T(R))*(sc_2/T(R))*\ldots(sc_n/T(R))]$$

$\sigma_{\theta_1\vee\theta_2\vee\ldots\vee\theta_n}(R)$ probability that a record satisfy none of θ:

$$[(1-sc_1/T(R))*(1-sc_2/T(R))*\ldots*(1-sc_n/T(R))]$$

$$T(R)*(1-[(1-sc_1/T(R))*(1-sc_2/T(R))*\ldots*(1-sc_n/T(R))])$$

Give the main steps of query optimization (diagram) (06_UW_query_proc 17.)



Give two conventional wisdom rules about query optimization (06_UW_query_proc 41.)

?

Conventional wisdom:
do projects early

Example: R(A,B,C,D,E) x={E}
P: (A=3) ∧ (B="cat")

$\pi_x\{\sigma_p(R)\}$ vs. $\pi_E\{\sigma_p\{\pi_{ABE}(R)\}\}$

Usually good: early selections

What is the difference between ALL_ROWS and FIRST_ROWS optimization modes? (07_tuning 6.)

Cost vs. Rule

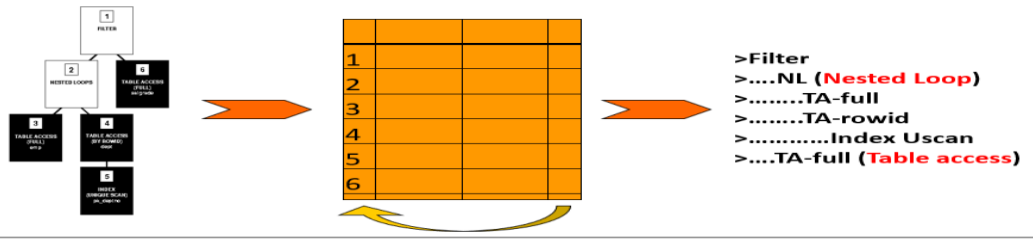
- Rule (RBO: Rule Based Optimization)
  - Hardcoded heuristic rules determine plan
    - "Access via index is better than full table scan"
    - "Fully matched index is better than partially matched index"
    - ...
- Cost (2 modes)
  - Statistics of data play role in plan determination
    - Best throughput mode: retrieve **all rows** asap
      - First compute, then return fast
    - Best response mode: retrieve **first row** asap
      - Start returning while computing (if possible)

What does an EXPLAIN PLAN statement do? (07_tuning 15.) -> stores plan in plan_table

What is stored in PLAN_TABLE? (07_tuning 15.) -> filter,nested loop,..?

Explain Plan Utility

- "Explain plan for <SQL-statement>"
  - Stores plan (row-sources + operations) in Plan_Table
  - View on Plan_Table (or 3rd party tool) formats into readable plan



What is a Full Table Scan operation (07_tuning 18.)

- Full table scan (FTS)
  - All blocks read sequentially into buffer cache
    - Also called "buffer-gets"
    - Done via multi-block I/O's (db_file_multiblock_read_count)
    - Till high-water-mark reached (truncate resets, delete not)
  - Per block: extract + return all rows
    - Then put block at LRU-end of LRU list (!)
    - All other operations put block at MRU-end

What is an Index Unique Scan operation (07_tuning 25.)

- Index Unique Scan

- Traverses the node blocks to locate correct leaf block
- Searches value in leaf block (if not found => done)
- Returns rowid to parent row-source
    - Parent: accesses the file+block and returns the row

What is an Index Range Scan operation (07_tuning 27.)

- (Non-unique) Index Range Scan
    - Traverses the node blocks to locate most left leaf block
    - Searches 1st occurrence of value in leaf block
    - Returns rowid to parent row-source
        - Parent: accesses the file+block and returns the row
    - Continues on to next occurrence of value in leaf block
        - Until no more occurences

What is Clustering Factor? (07_tuning 40.)

- Index level statistic
    - How well ordered are the rows in comparison to indexed values?
    - Average number of blocks to access a single value
        - 1 means range scans are cheap
        - <# of table blocks> means range scans are expensive
    - Used to rank multiple available range scans

What is the difference between Explain Plan and Tracing? (07_tuning 69.)

- Explain-plan: give insight before execution
- Tracing: give insight in actual execution
    - CPU-time spent
    - Elapsed-time

- # of physical block-I/O's
- # of cached block-I/O's
- Rows-processed per row-source

What can we do with hints? (07_tuning 72.)

- Force optimizer to pick specific alternative

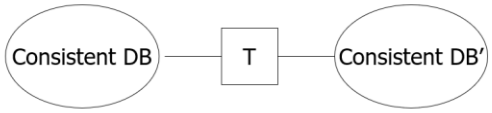
What can we do with ANALYZE command? (07_tuning 74)

- Statistics need to be periodically generated

What does it mean: a consistent database? (09_UW_crash_recovery 4.)

- Consistent DB: DB in consistent state

What is a transaction? (09_UW_crash_recovery 10.)



Transaction: collection of actions
that preserve consistency

How can constraints be violated? (09_UW_crash_recovery 12.)

How can constraints be violated?

- Transaction bug (incomplete transaction)
- DBMS bug (some process)
- Hardware failure

  e.g., disk crash alters balance of account
- Data sharing

  e.g.: T1: give 10% raise to programmers

  T2: change programmers ⇒ systems analysts

What are undesired expected events? (09_UW_crash_recovery 15.)

Desired events: see product manuals....

Undesired expected events:

System crash

- memory lost
- cpu halts, resets

that's it!!

Undesired Unexpected: Everything else!

Give the 3 important address spaces of a DBMS. (09_UW_crash_recovery 18.)

The primitive operations of Transactions

There are 3 important address spaces:

1. The disk blocks
2. The shared main memory
3. The local address space of a Transaction

Describe the following operations: read, write, input, output. (09_UW_crash_recovery 20.)

Operations:

- Input (x): block containing x → memory
- Output (x): block containing x → disk

- Read (x,t): do input(x) if necessary
  t ← value of x in block
- Write (x,t): do input(x) if necessary
  value of x in block ← t

What does "atomicity" property of a transaction mean? (09_UW_crash_recovery 27.)

- **Need atomicity: execute all actions of a transaction or none at all**

Describe UNDO logging rules. (09_UW_crash_recovery 38.)

Undo logging rules

(1) For every action generate undo log record (containing old value)

(2) Before x is modified on disk, log records pertaining to x must be on disk (write ahead logging: WAL)

(3) Before commit is flushed to log, all writes of transaction must be reflected on disk

Give the write order to disk in case of UNDO logging (09_UW_crash_recovery 39.)

Must write to disk in the following order
(UNDO LOG)

1. The log records indicating changed database elements.

2. The changed database elements themselves.

3. The COMMIT log record.

Give the recovery rules in case of UNDO logging. (09_UW_crash_recovery 43.)

Recovery rules: Undo logging

- For every Ti with <Ti, start> in log:
  - If <Ti,commit> or <Ti,abort> in log, do nothing
  - Else
    - For all <Ti, *X*, *v*> in log:
      - write (*X*, *v*)
      - output (*X*)
    - Write <Ti, abort> to log

☒IS THIS CORRECT??

Give the UNDO recovery steps if you see the following log records on disk ...

What happens when a failure occurs during recovery from UNDO log? (09_UW_crash_recovery 44.)

Recovery rules: Undo logging

(1) Let S = set of transactions with
<Ti, start> in log, but no
<Ti, commit> (or <Ti, abort>) record in log

(2) For each <Ti, X, v> in log,
in reverse order (latest → earliest) do:
- if Ti ∈ S then { - write (X, v)
- output (X)

(3) For each Ti ∈ S do
- write <Ti, abort> to log (plus FLUSH LOG)

What if failure during recovery?

No problem! ☞ Undo idempotent

Give the steps of a simple checkpoint in UNDO logging. (09_UW_crash_recovery 50.)

Checkpoint

• simple checkpoint

Periodically:
(1) Do not accept new transactions
(2) Wait until all transactions finish
(3) Flush all log records to disk (log)
(4) Flush all buffers to disk (DB) (do not discard buffers)
(5) Write "checkpoint" record on disk (log)
(6) Resume transaction processing

Give the steps of a non-quiescent checkpoint in UNDO logging. (09_UW_crash_recovery 51.)

Checkpoint

• non-quiescent checkpoint

1. Write log record <START CKPT(T1, ...Tk)>
   T1 ... Tk are active transactions, and flush log.

2. Wait until all Ti-s commit or abort, but don't prohibit other transactions from starting.

3. When all Ti-s have completed, write a log record <END CKPT> and flush the log.

To which point do we have to scan backwards in UNDO log if we use checkpoint?(09_UW_crash_recovery 53.)

Describe REDO logging rules. (09_UW_crash_recovery 59.)

Redo logging rules

(1) For every action, generate redo log record (containing new value)

(2) Before X is modified on disk (DB), all log records for transaction that modified X (including commit) must be on disk

(3) Flush log at commit

(4) Write END record after DB updates flushed to disk

Give the write order to disk in case of REDO logging (09_UW_crash_recovery 60.)

Must write to disk in the following order (REDO LOG)

1. The log records indicating changed database elements.

2. The COMMIT log record.

3. The changed database elements themselves.

In Undo it was 2<->3

Give the recovery rules in case of REDO logging. (09_UW_crash_recovery 64.)

Recovery rules: Redo logging

(1) Let S = set of transactions with <Ti, commit> (and no <Ti, end>) in log

(2) For each <Ti, X, v> in log, in forward order (earliest → latest) do:

- if Ti ∈ S then
  - Write(X, v)
  - Output(X)

(3) For each Ti ∈ S, write <Ti, end> to Log (plus FLUSH LOG)

Give the REDO recovery steps if you see the following log records on disk ...

Give the steps of a non-quiescent checkpoint in REDO logging. (09_UW_crash_recovery 67.)

Checkpoint

- non-quiescent checkpoint

1. Write log record <START CKPT(T1, …Tk)>
   T1 … Tk are active (uncommitted) transactions, and flush log.
2. Write to disk all database elements that were written to buffers but not yet to disk by transactions that had already committed when the <START CKPT> record was written to the log. (dirty buffers)
3. Write an <END CKPT> record to the log and flush the log.

To which point do we have to scan backwards in REDO log if we use checkpoint? (09_UW_crash_recovery 69.)

When we scan the log backwards

<START T1>
<T1, A, 5>
<START T2>
<COMMIT T1>
<T2, B, 10>
<START CKPT (T2)>
<T2, C, 15>
<START T3>
<T3, D, 20>
<END CKPT>
<COMMIT T2>
<COMMIT T3>

If we first meet an <END CKPT> record, we need to scan no further back than the earliest of <START Ti> records (among corresponding <START CKPT(T1, T2, … Tk)>.

If we first meet a record < START CKPT (T1, … , *Tk)>*, then the crash occurred during the checkpoint.

We cannot be sure that committed transactions prior to the start of this checkpoint had their changes written to disk.

Thus, we must search back to the previous <END CKPT> record, find its matching < START CKPT (T1, … , *Tk)>*.

When we scan the log backwards

<START CKPT (…)>
…
<END CKPT>
<START T1>
<T1, A, 5>
<START T2>
<COMMIT T1>
<T2, B, 10>
<START CKPT (T2)>
<T2, C, 15>
<START T3>
<T3, D, 20>

If we first meet an <END CKPT> record, we need to scan no further back than the earliest of <START Ti> records (among corresponding <START CKPT(T1, T2, … Tk)>.

If we first meet a record < START CKPT (T1, … , *Tk)>*, then the crash occurred during the checkpoint.

We cannot be sure that committed transactions prior to the start of this checkpoint had their changes written to disk.

Thus, we must search back to the previous <END CKPT> record, find its matching < START CKPT (T1, … , *Tk)>*.

What are the key drawbacks of UNDO log and REDO log? (70.)

Key drawbacks:

- *Undo logging:* need frequent disk writes
- *Redo logging:* need to keep all modified blocks in memory until commit

Describe UNDO/REDO logging rules. (09_UW_crash_recovery 72.)

Rules

- Page X can be flushed before or after Ti commit
- Log record flushed before corresponding updated page (WAL)
- Flush at commit (log only)

UR1 *Before modifying any database element X on disk because of changes* made by some transaction *T, it is necessary that the update record <T, X, v, w> appear on disk.*

UR2 *A <COMMIT T> record must be flushed to disk as soon as it appears* in the log.

Give the recovery rules in case of UNDO/REDO logging. (09_UW_crash_recovery 73.)

The undo/redo recovery policy is:

1. Redo all the committed transactions in the order earliest-first, and
2. Undo all the incomplete transactions in the order latest-first.

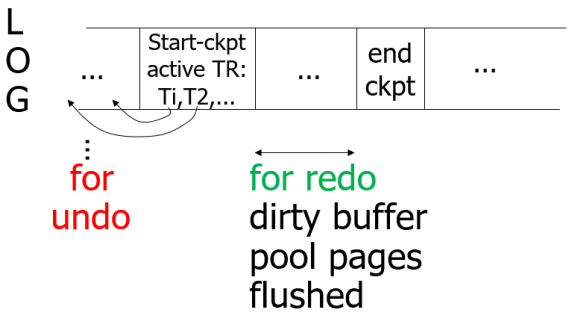
Give the steps of a non-quiescent checkpoint in UNDO/REDO logging. (09_UW_crash_recovery 75.)

Checkpoint

- non-quiescent checkpoint

1. Write log record <START CKPT(T1, ...Tk)>
   T1 ... Tk are active (uncommitted) transactions, and flush log.
2. Write to disk all the buffers that are *dirty;* i.e., they contain one or more changed database elements. Unlike redo logging, we flush all dirty buffers, not just those written by committed transactions.
3. Write a log record <END CKPT> and flush the log.

Non-quiescent checkpoint



What is concurrency control (10_UW_concurrency 3.)

Interactions among concurrently executing transactions can cause the database state to become inconsistent, even when the transactions individually preserve correctness of the state, and there is no system failure.

Thus, the timing of individual steps of different transactions needs to be regulated in some manner.

This regulation is the job of the *scheduler component of the DBMS, and the* general process of assuring that transactions preserve consistency when executing simultaneously is called *concurrency control.*

What is a schedule? (10_UW_concurrency 6., 14.)

A *schedule is a sequence of the important actions taken by one or more transactions.*

What is a serial schedule? (10_UW_concurrency 6.)

A schedule is *serial if its actions consist of all the actions of one transaction,* then all the actions of another transaction, and so on.

What is a serializable schedule? (10_UW_concurrency 9.)

A schedule *S is serializable if there is a serial*

schedule *S' such that for every initial database state, the effects of S and S'* are the same.

What does it mean: "conflict equivalent"? (10_UW_concurrency 23.)

S1, S2 are conflict equivalent schedules

if S1 can be transformed into S2 by a series of non-conflicting swaps of adjacent actions.

What does it mean: "conflict serializable"? (10_UW_concurrency 23.)

A schedule is conflict serializable if it is conflict equivalent to some serial schedule.

What is a precedence graph? (10_UW_concurrency 26.)

Precedence graph P(S) (S is schedule)

Nodes: transactions in S (T1, T2 …)

Arcs: Ti → Tj whenever

- $p_i(A)$, $q_j(A)$ are actions in S
- $p_i(A) <_S q_j(A)$ ($p_i(A)$ precedes $q_j(A)$ in S)
- at least one of p_i, q_j is a write

What can we say if two schedules are conflict equivalent? (10_UW_concurrency 29.)

Lemma

S_1, S_2 conflict equivalent $\Rightarrow P(S_1)=P(S_2)$

Proof:

Assume $P(S_1) \neq P(S_2)$

$\Rightarrow \exists\ T_i$: $T_i \rightarrow T_j$ in S_1 and not in S_2

$\Rightarrow S_1 = \ldots p_i(A) \ldots q_j(A) \ldots$

$S_2 = \ldots q_j(A) \ldots p_i(A) \ldots$

p_i, q_j conflict

$\Rightarrow S_1, S_2$ not conflict equivalent

Give two schedules whose precedence graphs are the same, but not confl. equivalent. (10_UW_concurrency 31.)

Note: $P(S_1)=P(S_2) \not\Rightarrow S_1, S_2$ conflict equivalent

Counter example:

$S_1 = w_1(A)\ r_2(A) \quad w_2(B)\ r_1(B)$ (cannot swap)

$S_2 = r_2(A)\ w_1(A) \quad r_1(B)\ w_2(B)$

30

Construct the precedence graph for the following schedule ...

What can we state about precedence graphs? (10_UW_concurrency 32.) ???

Theorem

$P(S_1)$ acyclic $\Leftrightarrow S_1$ conflict serializable

($\Leftarrow$) Assume S_1 is conflict serializable

$\Rightarrow \exists\ S_s$: S_s, S_1 conflict equivalent

$\Rightarrow P(S_s) = P(S_1)$

$\Rightarrow P(S_1)$ acyclic since $P(S_s)$ is acyclic

What does "consistency of a transaction" mean? (10_UW_concurrency 37.)

Rule #1: Consistency of transactions

T_i: ... $l_i(A)$ ... $p_i(A)$ ... $u_i(A)$ ...

1. A transaction can only read or write an element if it previously was granted a lock on that element and hasn't yet released the lock.
2. If a transaction locks an element, it must later unlock that element.

What does "legality of schedules" mean? (10_UW_concurrency 38.)

Rule #2 Legality of schedules

S = ... $l_i(A)$ ..... $u_i(A)$ .....

no $l_j(A)$

Locks must have their intended meaning: *no two* transactions may have locked the same element without one having first released the lock.

What does "two phase locking" mean? (10_UW_concurrency 42.)

Rule #3 Two phase locking (2PL)

for transactions

T_i = ... $l_i(A)$ ..... $u_i(A)$ .....

no unlocks no locks

What is a deadlock? How can we detect it? (10_UW_concurrency 51-54.)

A deadlock occurs when two or more sessions are waiting for data locked by each other, resulting in all the sessions being blocked. Oracle automatically detects and resolves deadlocks by rolling back the statement associated with the transaction that detects the deadlock.

Deadlocks

- Detection
  - Wait-for graph
- Prevention
  - Resource ordering
  - Timeout
  - Wait-die
  - Wound-wait

Deadlock Detection:

- Build Wait-For graph (Ti -> Tj edge if Ti waits for Tj)
- Use lock table structures
- Build incrementally or periodically
- When cycle found, rollback victim

What can we state about legal schedules of consistent 2PL transactions? (10_UW_concurrency 50.)

Theorem Rules #1,2,3 $\Rightarrow$ conflict serializable schedule

(consistency, legality, 2PL)

Intuitively, each two-phase-locked transaction may be thought to execute in its entirety at the instant it issues its first unlock request. In a conflict-equivalent serial schedule transactions appear in the same order as their first unlocks.

Give a serializable schedule which cannot be achieved via 2PL (10_UW_concurrency 58.)

???

What does "consistency of a transaction" mean in case of shared/exclusive locks? (10_UW_concurrency 63.)

Rule #1 Consistency of transactions

T_i =... l-S_1(A) ... r_1(A) ... u_1 (A) ...
T_i =... l-X_1(A) ... w_1(A) ... u_1 (A) ...

- A transaction may not write without holding an exclusive lock and may not read without holding some lock.
- All locks must be followed by an unlock of the same element.

What is legality of schedules in case of shared/exclusive locks? (10_UW_concurrency 64.)

Two-phase locking of transactions:

Locking must precede unlocking.

Legality of schedules:

An element may either be locked exclusively by one transaction or by several in shared mode, but not both.

What is the compatibility matrix? Give it for the shared/exclusive locking system. (10_UW_concurrency 68.)

Compatibility matrix

Comp

	S	X
S	true	false
X	false	false

Rows: held locks
Columns: requested locks

Give the compatibility matrix for shared/exclusive/increment locking system. (10_UW_concurrency 74.) (?)

Comp

	S	X	I
S	T	F	F
X	F	F	F
I	F	F	T

Legal schedule: defined by matrix

Any number of transactions can hold an increment lock on X at any time.

If an increment lock on X is held by some transaction, then no ther transaction can hold either a shared or exclusive lock on X at the same time.

Give the compatibility matrix for shared/exclusive/update locking system. (10_UW_concurrency 77.) ?

Comp

New request

Lock already held in

	S	X	U
S	T	F	T
X	F	F	F
U	F	F	F

-> not symmetric table

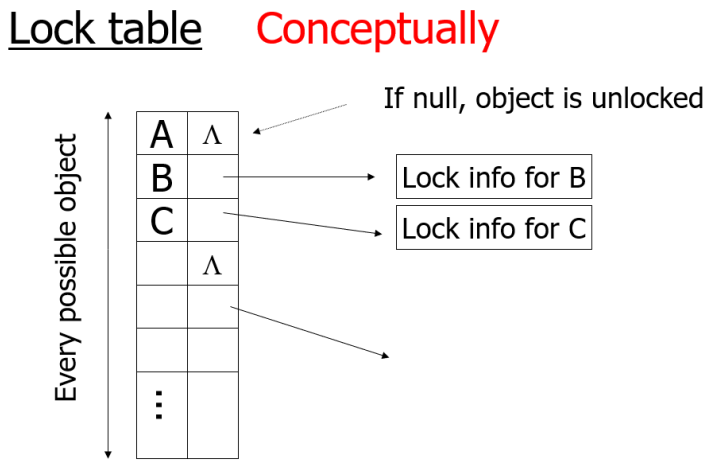
How DBMS-es guarantee 2PL in practice? (10_UW_concurrency 80.) -> group mode?

Note: object A may be locked in different modes at the same time...

S_1=...l-S_1(A)...l-S_2(A)...l-U_3(A)... { l-S_4(A)...? ; l-U_4(A)...?

- To grant a lock in mode t, mode t must be compatible with all currently held locks on object (-> we need a group mode)

What information can we find in Lock Tables of a DBMS? (10_UW_concurrency 82-84.)



What is the advantage/disadvantage of locking large objects? (10_UW_concurrency 87.)

What is the advantage/disadvantage of locking small objects? (10_UW_concurrency 87.)

- Locking works in any case, but should we choose small or large objects?

- If we lock large objects (e.g., Relations)
  - Need few locks
  - Low concurrency
- If we lock small objects (e.g., tuples,fields)
  - Need more locks
  - More concurrency