UNIVERSITAS INDONESIA

FAKULTAS ILMU KOMPUTER

14

Physical Design

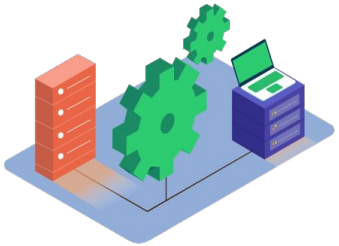
CSF2600700 - BASIS DATA

Acknowledgements

This slide is a modification to a slide titled “**Physical Database Design**”used in “Basis Data” course in academic years 2018/2019 in the Faculty of Computer Science, Universitas Indonesia

Objectives

Understand **principles** of a good physical database design.



Able to **implement** physical database design

Outline

1. Logical VS Physical Database Design

2. Physical Database Design Methodology

Logical vs Physical Database Design

Sources of information for physical design process includes **logical data model and documentation** that describes model.

- ➔ Logical database design is concerned with the **what**
- ➔ Physical database design is concerned with the **how**

Physical Database Design

Process of producing a description of the implementation of the database on secondary storage.

It describes the **base relations**, **file organizations**, and **indexes** used to achieve efficient access to the data, and any associated **integrity constraints** and **security measures**.

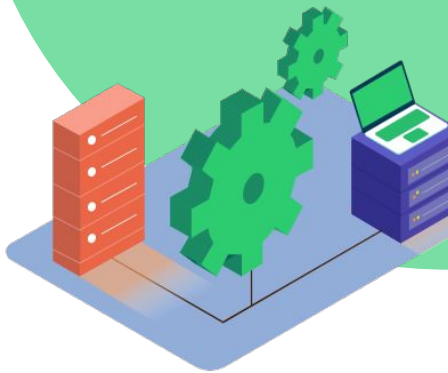
Physical Database Design (Cntd.)

Purpose - translate the logical description of data into the technical specifications for storing and retrieving data

Goal - create a design for storing data that will provide adequate performance and insure database integrity, security and recoverability

Physical design describes the **base relations**, **file organizations**, and **indexes** used to achieve efficient access to the data, and any associated integrity constraints and security measures.

Physical Design Process



Inputs	Decision
➔ Normalized relations	➔ Attribute data types
➔ Volume estimates	➔ Physical record descriptions (doesn't always match logical design)
➔ Attribute definitions	➔ File organizations
➔ Response time expectations	➔ Indexes and database architectures Query optimization
➔ Data security needs Backup/recovery needs	
➔ Integrity expectations	
➔ DBMS technology used	

Physical Database Design Methodology

1. Translate logical data model for target DBMS
- ➔ Design base relations
- ➔ Design representation of derived data
- ➔ Design general constraints

2. Consider for Denormalization

3. Consider for Partitioning

4. Design file organizations and indexes

5. Design user views

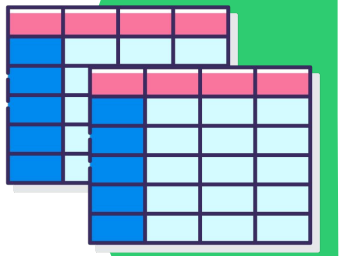
6. Design security mechanisms

7. Monitor and tune operational system

8. Consider for RAID

STEP 1 - Translate Logical Data Model for Target DBMS

To produce a **relational database schema** from the logical data model that can be implemented in the target DBMS.

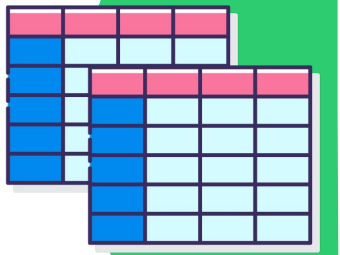


Need to know functionality of target DBMS such as how to create base relations and whether the system supports the definition of:

- ➔ PKs, FKs, and AKs;
- ➔ required data – i.e. whether system supports NOT NULL;
- ➔ domains;
- ➔ relational integrity constraints;
- ➔ general constraints.

1.1 - Design base relations

To decide **how to represent base relations** identified in logical model in target DBMS.



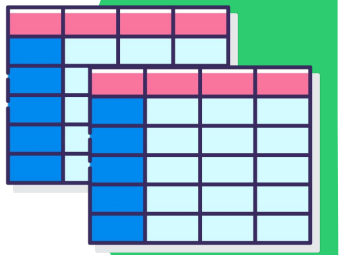
For each relation, need to define:

- ➔ the name of the relation;
- ➔ a list of simple attributes in brackets;
- ➔ the PK and, where appropriate, AKs and FKs.
- ➔ integrity constraints:
- ➔ Null value control – allowing or prohibiting empty fields
- ➔ Referential integrity – range control (and null value allowances) for foreign-key to primary-key match-ups

1.1 - Design base relations (Cntd.)

From data dictionary, we have for each attribute:

- ➔ its **domain**, consisting of a data type, length, and any constraints on the domain;
- ➔ an **optional default value** for the attribute;
- ➔ whether it can **hold nulls**;
- ➔ whether it is **derived**, and if so, how it should be computed.



DBDL for the PropertyForRent Relation

```
    Domain PropertyNumber:          variable length character string, length 5
    Domain Street:                  variable length character string, length 25
    Domain City:                    variable length character string, length 15
    Domain Postcode:                variable length character string, length 8
    Domain PropertyType:            single character, must be one of 'B', 'C', 'D', 'E', 'F', 'H', 'M', 'S'
    Domain PropertyRooms:           integer, in the range 1–15
    Domain PropertyRent:            monetary value, in the range 0.00–9999.99
    Domain OwnerNumber:             variable length character string, length 5
    Domain StaffNumber:             variable length character string, length 5
    Domain BranchNumber:            fixed length character string, length 4
PropertyForRent(
    propertyNo      PropertyNumber      NOT NULL,
    street          Street              NOT NULL,
    city            City                NOT NULL,
    postcode        Postcode,
    type            PropertyType        NOT NULL DEFAULT 'F',
    rooms           PropertyRooms       NOT NULL DEFAULT 4,
    rent            PropertyRent        NOT NULL DEFAULT 600,
    ownerNo         OwnerNumber         NOT NULL,
    staffNo         StaffNumber,
    branchNo        BranchNumber        NOT NULL,
    PRIMARY KEY (propertyNo),
    FOREIGN KEY (staffNo) REFERENCES Staff(staffNo) ON UPDATE CASCADE ON DELETE SET NULL,
    FOREIGN KEY (ownerNo) REFERENCES PrivateOwner(ownerNo) and BusinessOwner(ownerNo)
                                    ON UPDATE CASCADE ON DELETE NO ACTION,
    FOREIGN KEY (branchNo) REFERENCES Branch(branchNo)
                                    ON UPDATE CASCADE ON DELETE NO ACTION);
```

1.2 - Design representation of derived data

To decide how to represent any **derived** data present in logical data model in target DBMS.

- ➔ Examine logical data model and data dictionary, and produce list of all derived attributes.
- ➔ Derived attribute **can be stored** in database or **calculated every time** it is needed.

1.2 - Design representation of derived data (Cntd.)

Option selected is based on:

1. **additional cost to store** the derived data and keep it consistent with operational data from which it is derived;

2. **cost to calculate** it each time it is required.

Less expensive option is chosen subject to performance constraints.

PropertyforRent Relation and Staff Relation with Derived Attribute noOfProperties

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003

Staff

staffNo	fName	lName	branchNo	noOfProperties
SL21	John	White	B005	0
SG37	Ann	Beech	B003	2
SG14	David	Ford	B003	1
SA9	Mary	Howe	B007	1
SG5	Susan	Brand	B003	0
SL41	Julie	Lee	B005	1

1.3 - Design general constraints

To **design the general constraints** for target DBMS.

Some DBMS provide more facilities than others for defining enterprise constraints.

Example:

```
CONSTRAINT StaffNotHandlingTooMuch
CHECK (NOT EXISTS (SELECT staffNo
                   FROM PropertyForRent
                   GROUP BY staffNo
                   HAVING COUNT(*) > 100));
```

STEP 2 - Consider for Denormalization

Transforming normalized relations into unnormalized physical record specifications

Benefits:
Can improve performance (speed) be reducing number of table lookups (i.e reduce number of necessary join queries)

Costs (due to data duplication)
- ➔ Wasted storage space
- ➔ Data integrity/consistency threats

We have discussed this in Tuning and Monitoring

STEP 3 - Consider for Partitioning

Horizontal Partitioning:
Distributing the rows of a table into several separate files

- ➔ Useful for situations where different users need access to different rows
- ➔ Three types: Key Range Partitioning, Hash Partitioning, or Composite Partitioning

Vertical Partitioning:
Distributing the columns of a table into several separate files ◦

- ➔ Useful for situations where different users need access to different columns
- ➔ The primary key must be repeated in each file

Combinations of Horizontal and Vertical Partitions often correspond with User Schemas (user views)

STEP 3 - Consider for Partitioning (Cntd.)

Advantages of Partitioning

- ➔ **Efficiency**: Records used together are grouped together
- ➔ **Local optimization**: Each partition can be optimized for performance
- ➔ **Security**, recovery
- ➔ **Load balancing**: Partitions stored on different disks, reduces contention
- ➔ Take advantage of **parallel processing** capability

Disadvantages of Partitioning:

- ➔ Inconsistent access speed: **Slow** retrievals across partitions
- ➔ **Complexity**: non-transparent partitioning
- ➔ **Extra space** or update time: duplicate data; access from multiple partitions

STEP 4 - Design File Organizations and Indexes

To determine **optimal file organizations** to store the base relations and the **indexes** that are required to achieve acceptable performance; that is, the way in which relations and tuples will be held on secondary storage.

Must understand the typical **workload** that database must support.

4.1 - Analyze transactions

To understand the **functionality of the transactions** that will run on the database and to **analyze the important transactions**

Attempt to identify performance criteria, such as:

- ➔ transactions that run frequently and will have a significant impact on performance; ◦
- ➔ transactions that are critical to the business; ◦
- ➔ times during the day/week when there will be a high demand made on the database (called the peak load).

Use this information to identify the parts of the database that may cause performance problems.

Also need to know high-level functionality of the transactions, such as:

- ➔ attributes that are updated;
- ➔ search criteria used in a query

4.1 - Analyze transactions (Cntd.)

Often not possible to analyze all transactions, so investigate most ‘important’ ones. To focus on areas that may be problematic:

(1) **Map all transaction paths** to relations.

(2) Determine which relations are most **frequently** accessed by transactions.

(3) Analyze the **data usage** of selected transactions that involve these relations.

Cross-referencing transactions and relations

Table 17.1 Cross-referencing transactions and relations.

Transaction/ Relation	(A)				(B)				(C)				(D)				(E)				(F)			
	I	R	U	D	I	R	U	D	I	R	U	D	I	R	U	D	I	R	U	D	I	R	U	D
Branch										X				X								X		
Telephone																								
Staff		X				X				X								X				X		
Manager																								
PrivateOwner	X																							
BusinessOwner	X																							
PropertyForRent	X					X	X	X						X				X				X		
Viewing																								
Client																								
Registration																								
Lease																								
Newspaper																								
Advert																								

I = Insert; R = Read; U = Update; D = Delete

For example transaction A: enter the details for new property and the owner (such as details of property number PG4 in Glasgow owned by Tina Murphy)

4.2 - Choose file organizations

Technique for physically arranging records of a file on secondary storage

Types of file organizations

Sequential | **Indexed** | **Hashed**

4.2 - Choose file organizations (Cntd.)

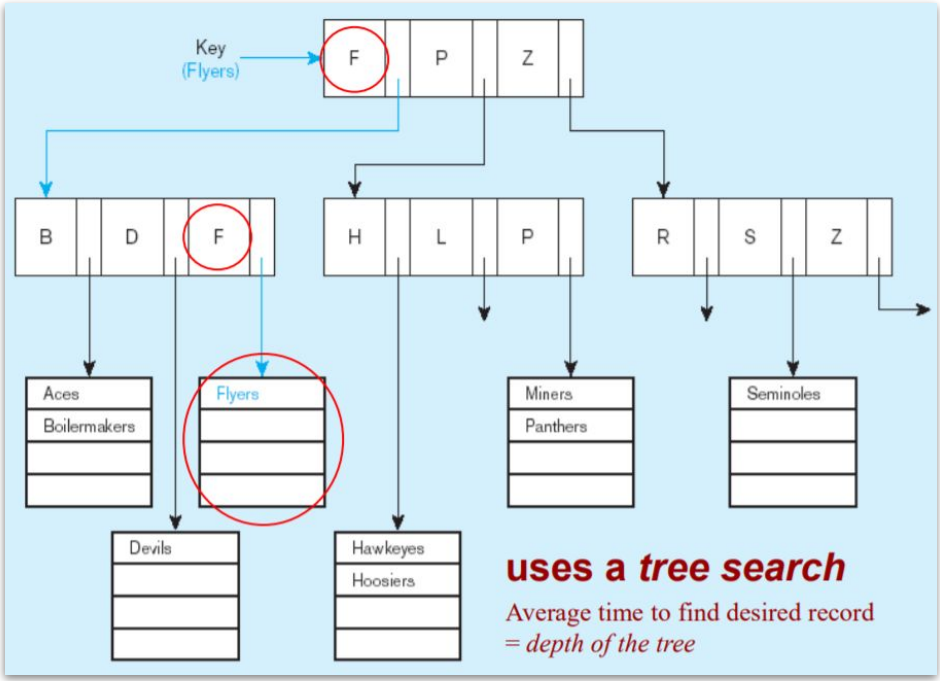
Indexed File Organizations

Index – a separate table that contains organization of records for quick retrieval
Primary keys are automatically indexed

Indexing approaches:

- **B-tree index**
- **Hash Index**
- **Bitmap index**
- **Join Index**

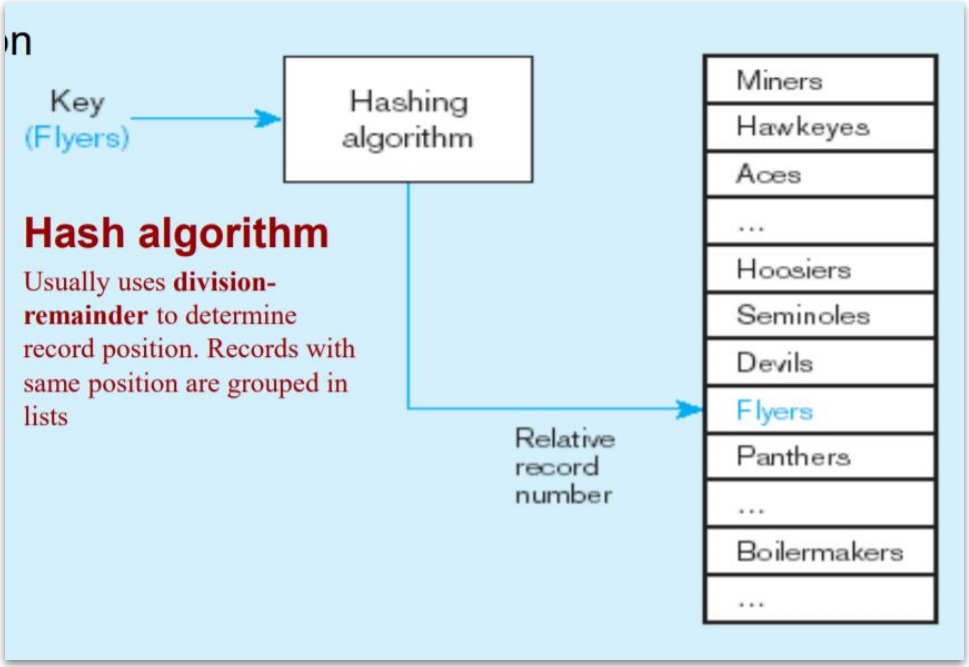
4.2 - Choose file organizations (Cntd.)



B-tree Index

Leaves of the tree are all at same level
➔ **consistent access time**

4.2 - Choose file organizations (Cntd.)



Hashed file or
index organization

4.2 - Choose file organizations (Cntd.)

	Product Table Row Numbers									
Price	1	2	3	4	5	6	7	8	9	10
100	0	0	1	0	1	0	0	0	0	0
200	1	0	0	0	0	0	0	0	0	0
300	0	1	0	0	0	0	1	0	0	1
400	0	0	0	1	0	1	0	1	1	0

Products 3 and 5 have Price $100
Product 1 has Price $200
Products 2, 7, and 10 have Price $300
Products 4, 6, 8, and 9 have Price $400

Bitmap index organization

Bitmap saves on space requirements

Rows - possible values of the attribute

Columns - table rows

Bit - indicates whether the attribute of a row has the values

4.2 - Choose file organizations (Cntd.)

Join Indexes – speeds up join operations

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
. . .		

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	C1026	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
. . .		

4.3 - Rules for Using Index

Indexing mechanisms used to **speed up** access to desired data.

An index is like a card catalog in a library. Each card (entry) has: (keyvalue, call code)

- ➔ keyvalue is for lookup (title or author),
- ➔ call code is used to locate the actual location of the book.

Entries are placed in alphabetical order by lookup key. (why?)

Index is **ACCESS PATH** to records

4.3 - Rules for Using Index (Cntd.)

Some rules

1. Use on **larger tables**
2. **Index the primary key** of each table
3. Index search fields (fields frequently in **WHERE** clause)
4. Fields in SQL **ORDER BY** and **GROUP BY** commands
5. When there are >100 values but not when there are <30 values

6. **Avoid use of indexes for fields with long values**; perhaps compress values first
7. **DBMS may have limit** on number of indexes per table and number of bytes per indexed field(s)
8. **Null values** will not be referenced from an index
9. Use indexes heavily for non-volatile databases; **limit the use of indexes for volatile** databases. Why? Because modifications (e.g. inserts, deletes) require updates to occur in index files

STEP 5 - Design User Views

To design the user views that were identified during the Requirements Collection and Analysis stage of the database system development lifecycle.

CUSTOMER

Name	Addr	Sex	Age

Base table

MALE_CUSTOMER (Name, Addr) =
CUSTOMER (Name, Addr, Sex, Age)
WHERE Sex = M

User view

MALE_CUSTOMER

Name	Addr

STEP 6 - Design Security Measures

To design the security measures for the database as specified by the users.

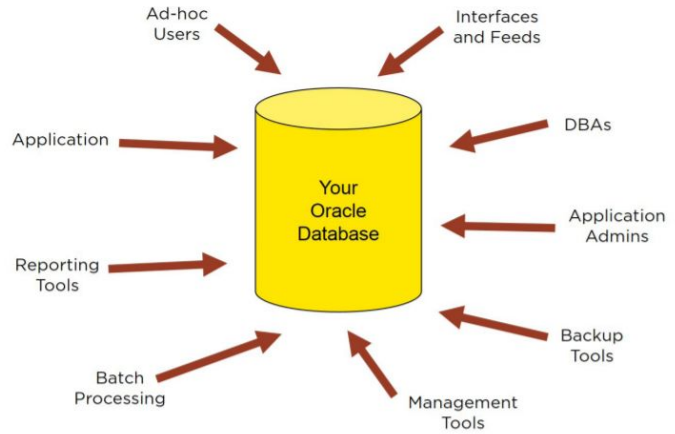


Figure 1: Database security is a complex problem.

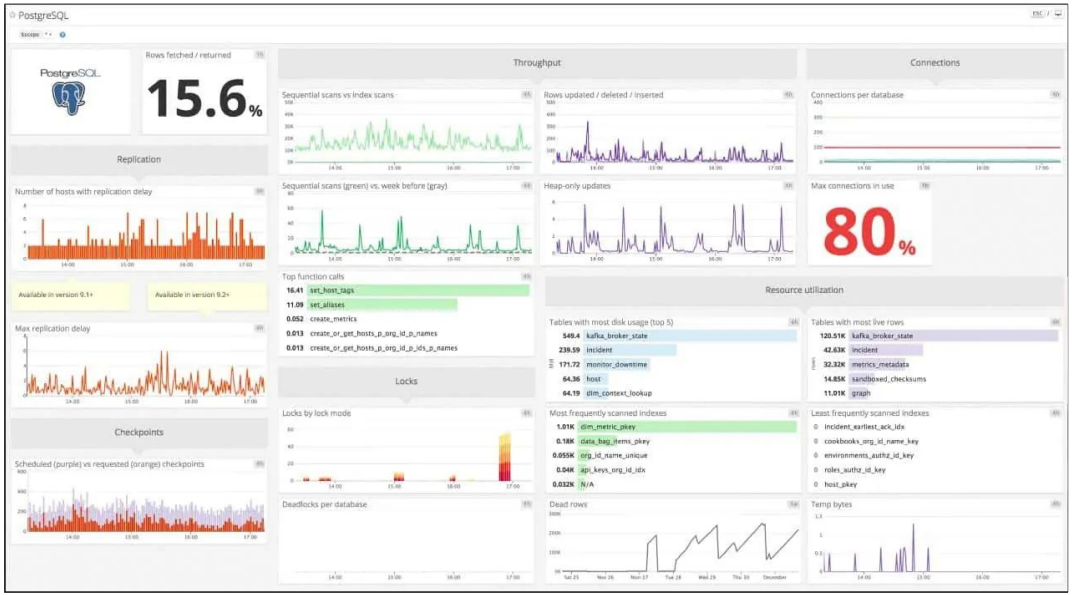
Inventory	▪ An inventory of all databases and sensitive data locations. Once you know where your sensitive data is, you can consider removing it from scope, period; removing it via P2PE; or protecting it with the other options presented in this white paper. ▪ Methods and processes to maintain the inventories
Configuration	▪ A measureable database security standard and baseline ▪ Periodic validation with compliance to the standard
Access	▪ Database access management policies, procedures, and tools ▪ Database access profiling and monitoring
Auditing	▪ Database auditing requirements, processes, and definitions ▪ Centralized auditing retention and reporting solution
Monitoring	▪ Database real-time security monitoring and intrusion detection ▪ Database monitoring definition and tools
Vulnerability	▪ Vulnerability assessment and management for databases ▪ Vulnerability remediation strategy and processes
Protection	▪ Sensitive data protection strategy – encryption, data masking, redaction, scrambling, tokenization, P2PE. ▪ Data protection policies, procedures, and tools

Table 2: Database security program components

Source: Rimini Street (2016). Understanding Comprehensive Database Security

STEP 7 - Monitoring and Tuning

Monitoring **database performance**



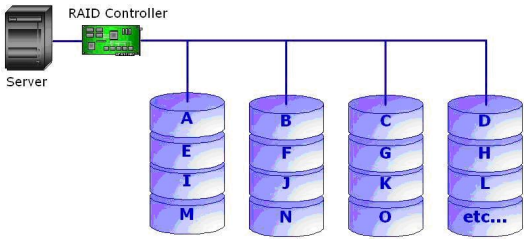
An example of **monitoring tool** for Postgres DB: *Datadog APM Module*

https://www.comparitech.com/net-admin/best-postgresql-monitoring-tools/

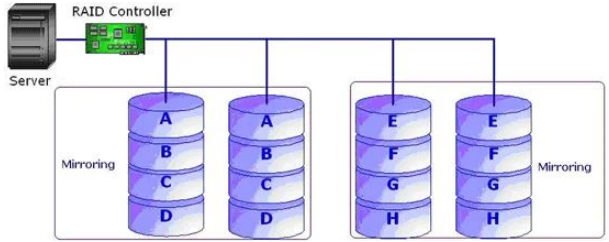
STEP 8 - Consider for RAID

- ➔ **Redundant Array of Inexpensive Disks (RAID)**
- ➔ **A set of disk drives** that appear to the user to be a single disk drive
- ➔ Allows **parallel access** to data (improves access speed)
- ➔ Pages are **arranged in stripes**

Examples:



LEVEL 0



LEVEL 1

https://www.howtoforge.com/redundant-array-of-inexpensive-disks-raid-technical-paper

Q&A