Benefits of Database Technology

1. High-level/declarative DML (query) Languages

Physical Data Independence

Ability to develop/change the physical schema without changing the conceptual (logical) schema.

⇒ essential to fully realize productivity gains in development.

2. Transactions and Concurrency Control

3. Recovery

4. . . .
Benefits of Database Technology

1. High-level/declarative DML (query) Languages

   **Physical Data Independence**

   Ability to develop/change the physical schema without changing the conceptual (logical) schema.

   ⇒ essential to fully realize *productivity gains* in development.

2. Transactions and Concurrency Control

3. Recovery

4. . . .
Benefits of Database Technology

1. High-level/declarative DML (query) Languages

Physical Data Independence

Ability to develop/change the physical schema without changing the conceptual (logical) schema.

⇒ essential to fully realize *productivity gains* in development.

2. Transactions and Concurrency Control

3. Recovery

4. . . .
Conceptual/Logical vs. Physical Data

IDEA:
Insulate Users/Applications from Physical Design issues ... essentially ADTs for DATA

Issues:
- DDL Languages?
- How are the schemata linked together?
- How to execute DML requests?
Conceptual/Logical vs. Physical Data

IDEA:
Insulate Users/Applications from Physical Design issues . . . essentially ADTs for DATA

Issues:
- DDL Languages?
- How are the schemata linked together?
- How to execute DML requests?
Queries vs. Query Processing

(1) User/App Queries:
- formulated w.r.t. the conceptual/external schema
- high level (declarative) query languages (SQL, OQL)
  
  logic-based semantics based on satisfaction
  “does a database $D$ and a tuple $t$ make a query $Q$ true?”

(2) Query PLANS:
- formulated w.r.t. physical schema
- low-level iterator-based language (relational algebra)

Problem:
How to get from (1) to (2)? Query Optimization!
Queries vs. Query Processing

(1) User/App Queries:
- formulated w.r.t. the conceptual/external schema
- high level (declarative) query languages (SQL, OQL)

**logic-based semantics based on satisfaction**
“does a database \( D \) and a tuple \( t \) make a query \( Q \) true?”

(2) Query PLANS:
- formulated w.r.t. physical schema
- low-level iterator-based language (relational algebra)

Problem:
How to get from (1) to (2)?
Queries vs. Query Processing

(1) User/App Queries:

* formulated w.r.t. the conceptual/external schema
* high level (declarative) query languages (SQL, OQL)

**logic-based semantics based on satisfaction**

“does a database $D$ and a tuple $t$ make a query $Q$ true?”

(2) Query PLANS:

* formulated w.r.t. physical schema
* low-level iterator-based language (relational algebra)

Problem:

How to get from (1) to (2)?

Query Optimization!
First (BAD) IDEA:

Logical schema is closely tied to Physical Schema

... this simplifies *Query Optimization* (hence mostly focus on costs)

Example (RDBMS)

The DDL statement `CREATE TABLE`:

1. declares a *new relation* (conceptual; includes keys, ...)
2. creates a *base file* (physical; includes structure, placement, ...)

This approach has been followed for \( \sim 30 \) years

... the IBM DB2’s `CREATE TABLE` now has *3 pages of options*

But there are other related DDL statements:

... `CREATE VIEW` (conceptual) and `CREATE INDEX` (physical)
Standard Approach to Physical Design (and Queries)

First (BAD) IDEA:

Logical schema is closely tied to Physical Schema

... this simplifies Query Optimization (hence mostly focus on costs)

Example (RDBMS)

The DDL statement `CREATE TABLE`:

1. declares a new relation (conceptual; includes keys, ...)
2. creates a base file (physical; includes structure, placement, ...)

This approach has been followed for \( \sim 30 \) years

... the IBM DB2’s `CREATE TABLE` now has 3 pages of options.

But there are other related DDL statements:

... `CREATE VIEW` (conceptual) and `CREATE INDEX` (physical)
First (BAD) IDEA:

Logical schema is closely tied to Physical Schema

... this simplifies Query Optimization (hence mostly focus on costs)

Example (RDBMS)

The DDL statement CREATE TABLE:

1. declares a new relation (conceptual; includes keys, ...)
2. creates a base file (physical; includes structure, placement, ...)

This approach has been followed for ~ 30 years

... the IBM DB2’s CREATE TABLE now has 3 pages of options.

But there are other related DDL statements:

... CREATE VIEW (conceptual) and CREATE INDEX (physical)
Common Advice to “DB tuning”

Second (BAD, but practical and expedient) IDEA:

(In closely-tied approaches) Physical Design can be greatly influenced by changes to the Conceptual/Logical Schema. ... hence we don’t have to change query optimizer.

Example (in RDBMS)

1. Denormalization (and NULLs)
2. Horizontal/Vertical Partitioning
Common Advice to “DB tuning”

Second (BAD, but practical and expedient) IDEA:
(In closely-tied approaches) Physical Design can be *greatly influenced* by *changes to the Conceptual/Logical Schema*

... hence we don’t have to change query optimizer

Example (in RDBMS)

1. Denormalization (and NULLs)
2. Horizontal/Vertical Partitioning
Denormalization

Golden Rule for Conceptual to Logical Mappings

Independent Entities (Relationships) are kept separate

⇒ normal forms (relational: BCNF, 3NF)

... makes “complex object reconstruction” harder (joins)

ADVICE:

Settle for lower Normal Form
(i.e., combine multiple entities/relationships into one logical unit).

... avoids joins

PROBLEMS:

- update anomalies (often leads to proliferation of NULLs!)
- increase of storage space (in the standard approach)
- queries/updates have to be reformulated
Denormalization

Golden Rule for Conceptual to Logical Mappings

Independent Entities (Relationships) are kept separate

$\Rightarrow$ normal forms (relational: BCNF, 3NF)

$\ldots$ makes “complex object reconstruction” harder (joins)

ADVICE:

Settle for lower Normal Form
(i.e., combine multiple entities/relationships into one logical unit).

$\ldots$ avoids joins

PROBLEMS:

- update anomalies (often leads to proliferation of NULLs!)
- increase of storage space (in the standard approach)
- queries/updates have to be reformulated
Denormalization

Golden Rule for Conceptual to Logical Mappings

Independent Entities (Relationships) are kept separate

⇒ normal forms (relational: BCNF, 3NF)

... makes “complex object reconstruction” harder (joins)

ADVICE:

Settle for lower Normal Form
(i.e., combine multiple entities/relationships into one logical unit).

... avoids joins

PROBLEMS:

- update anomalies (often leads to proliferation of NULLs!)
- increase of storage space (in the standard approach)
- queries/updates have to be reformulated
Denormalization

Golden Rule for Conceptual to Logical Mappings

Independent Entities (Relationships) are kept separate

⇒ normal forms (relational: BCNF, 3NF)

... makes “complex object reconstruction” harder (joins)

ADVICE:

Settle for lower Normal Form
(i.e., combine multiple entities/relationships into one logical unit).

... avoids joins

PROBLEMS:

• update anomalies (often leads to proliferation of NULLs!)
• increase of storage space (in the standard approach)
• queries/updates have to be reformulated
Denormalization: Example

Normalized Design:

```sql
employee(id, name, dept)
department(dept, address)
```

Denormalization:

```sql
empdept(id, name, dept, address)
```
ADVICE:
Make *logical units* correspond to frequent requests:

**Vertical Partition:**
- attributes together in a query form fragments

**Horizontal Partition:**
- value ranges in queries form fragments

PROBLEMS:
- lossless vs. efficient designs
- queries/updates have to be reformulated
ADVICE:
Make *logical units* correspond to frequent requests:

**Vertical Partition:**
- attributes together in a query form fragments

**Horizontal Partition:**
- value ranges in queries form fragments

PROBLEMS:
- lossless vs. efficient designs
- queries/updates have to be reformulated
Problems?

Second (BAD, but practical and expedient) IDEA:
(In closely-tied approaches) Physical Design can be
*greatly influenced* by *changes to the Conceptual/Logical Schema*

... completely breaks *Physical Data Independence*

- Materialized Views (essentially additional tables)
- Data Cubes (summary tables)

Physical design $\sim$ changes to logical schema + index selection
Problems?

Second (BAD, but practical and expedient) IDEA:

(In closely-tied approaches) Physical Design can be greatly influenced by changes to the Conceptual/Logical Schema.

... completely breaks Physical Data Independence.

- Materialized Views (essentially additional tables)
- Data Cubes (summary tables)

... while preserving Data Integrity?

Physical design ~ changes to logical schema + index selection
Problems?

Second (BAD, but practical and expedient) IDEA:

(In closely-tied approaches) Physical Design can be greatly influenced by changes to the Conceptual/Logical Schema.

... completely breaks Physical Data Independence.

- Materialized Views (essentially additional tables)
- Data Cubes (summary tables)

... while preserving Data Integrity?

Physical design $\sim$ changes to logical schema + index selection
Built-in Assumptions: 2-level Storage and Clustering

Additional complications:

- implicit assumption of two-level storage
  
  ... most current DBMS assume this
  
  ... specialized main-memory DBMS

- impact on data structures (indices)
  
  ... primary vs. secondary indices
  
  ... clustered vs. unclustered indices
Built-in Assumptions: 2-level Storage and Clustering

Additional complications:

- implicit assumption of two-level storage
  - most current DBMS assume this
  - specialized main-memory DBMS

- impact on data structures (indices)
  - primary vs. secondary indices
  - clustered vs. unclustered indices
Goals and Solutions

GOAL 1:
Develop approaches and technology that allow for loose coupling between conceptual/logical designs and physical design.

... allows logical design to closely follow the conceptual design

... while supporting a wide variety of physical designs

GOAL 2:
Design a small number of primitives that support all of the above.

1. uniform DDL for both conceptual/logical and physical objects
2. capabilities (index) declarations for physical objects
3. integrity constraints to establish links between objects
4. no built-in assumptions (e.g., 2-level store)
Goals and Solutions

GOAL 1:
Develop approaches and technology that allow for loose coupling between conceptual/logical designs and physical design.

... allows logical design to closely follow the conceptual design
... while supporting a wide variety of physical designs

GOAL 2:
Design a small number of primitives that support all of the above.

1. uniform DDL for both conceptual/logical and physical objects
2. capabilities (index) declarations for physical objects
3. integrity constraints to establish links between objects
4. no built-in assumptions (e.g., 2-level store)
Organization of the Lectures

1. Uniform Approach to Physical Design and Schema Languages
2. How do we execute queries? (take 1: conjunctive queries)
3. How do we execute queries? (take 2: first-order queries)
4. Look into the Future (discussion/seminar)