1 Database

Data and Information

Data: facts stored and recorded, text, numbers, dates

Information: data presented in context, proceed to increase knowledge

Metadata: data about data. structure, rules, and constraints about data. ensures consistency. usernames, date, duration

DBMS

Database: large, integrated, structured collection of data. Models enterprise data. Has entities and relationships

DBMS: software system designed to store, maintain, facilitate access to databases

Database systems: manages data in a structured and relational way. Ethical issues of database breaches, keep personal identifiable info private and anonymize info when publishing

Problems and Benefits against file based

- Program-Data Dependence. File structure changes implies program changes; Data independence and central repository
- Duplication of data, loss of data integrity; Minimal data redundancy, improved data consistency.
- Limited data sharing, tied to application; Improved sharing
- Lengthy dev time, low level data management and file format management each time
- Excessive program maintenance, high percentage of dev time in maintaining file based; Reduced maintenance, data structure can change without changing data
- ;Novel data access using SQL

2 System Development

Dev lifecycle

Database planning: enterprise data model

Systems definition: scope, boundaries, interfacing with other systems

Requirements definition and analysis: collecting and analyze requirements

Database Design: Conceptual, Logical, Physical design

Conceptual Create data model independent of all physical considerations or logical models

Logical Construct relational model based on conceptual

Physical Description of implementation of logical model for a specific DBMS brand. Contains data types, file organization, indexes

Application Design: create interface and application using db

Implementation: Realization of database

Data Conversion and Loading : load existing data to db

Testing: tests for error, logical problems, performance, robustness, recoverability, adaptability

Operational Maintenance: monitor and maintaining after commission

Data types help DBMS to store and use information efficiently. It makes assumptions in computations, guarantee efficiency, minimize storage space, and help data integrity

CHAR(size) A FIXED length string. The size parameter specifies the column length in characters - can be from 0 to 255. Default is 1

VARCHAR(size) A VARIABLE length string. The size parameter specifies the maximum column length in characters - can be from 0 to 65535

BINARY(size) Equal to CHAR(), but stores binary byte strings. The size parameter specifies the column length in bytes. Default is 1

VARBINARY(size) Equal to VARCHAR(), but stores binary byte strings. The size parameter specifies the maximum column length in bytes.

TINYBLOB For BLOBs (Binary Large OBjects). Max length: 255 bytes

TINYTEXT Holds a string with a maximum length of 255 characters

TEXT(size) Holds a string with a maximum length of 65,535 bytes

BLOB(size) For BLOBs (Binary Large OBjects). Holds up to 65,535 bytes of data

MEDIUMTEXT Holds a string with a maximum length of 16,777,215 characters

LONGBLOB For BLOBs (Binary Large OBjects). Holds up to 4,294,967,295 bytes of data

ENUM(val1, val2, val3, ...) You can list up to 65535 values in an ENUM list.

SET(val1, val2, val3, ...) You can list up to 64 values in a SET list

BIT(size) A bit-value type. The number of bits per value is specified in size. The size parameter can hold a value from 1 to 64. The default value for size is 1.

TINYINT(size) Signed range is from -128 to 127. Unsigned range is from 0 to 255. The size parameter specifies the maximum display width (which is 255)

BOOL Zero is considered as false, nonzero values are considered as true.

BOOLEAN Equal to BOOL

SMALLINT(size) Signed range is from -32768 to 32767. Unsigned range is from 0 to 65535. The size parameter specifies the maximum display width (which is 255)

MEDIUMINT(size) Signed range is from -8388608 to 8388607. Unsigned range is from 0 to 16777215. The size parameter specifies the maximum display width (which is 255)

INT(size) Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The size parameter specifies the maximum display width (which is 255)

INTEGER(size) Equal to INT(size)

BIGINT(size) Signed range is from -9223372036854775808 to 9223372036854775807. Unsigned range is from 0 to 18446744073709551615. The size parameter specifies the maximum display width (which is 255)

FLOAT(size, d) The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter.

FLOAT(p) A floating point number. MySQL uses the p value to determine whether to use FLOAT or DOUBLE for the resulting data type. If p is from 0 to 24, the data type becomes FLOAT(). If p is from 25 to 53, the data type becomes DOUBLE()

DOUBLE(size, d) A normal-size floating point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter

DOUBLE PRECISION(size, d)

DECIMAL(size, d) An exact fixed-point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter. The maximum number for size is 65. The maximum number for d is 30. The default value for size is 10. The default value for d is 0.

DEC(size, d) Equal to DECIMAL(size,d)

DATE A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'

DATETIME(fsp) A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.

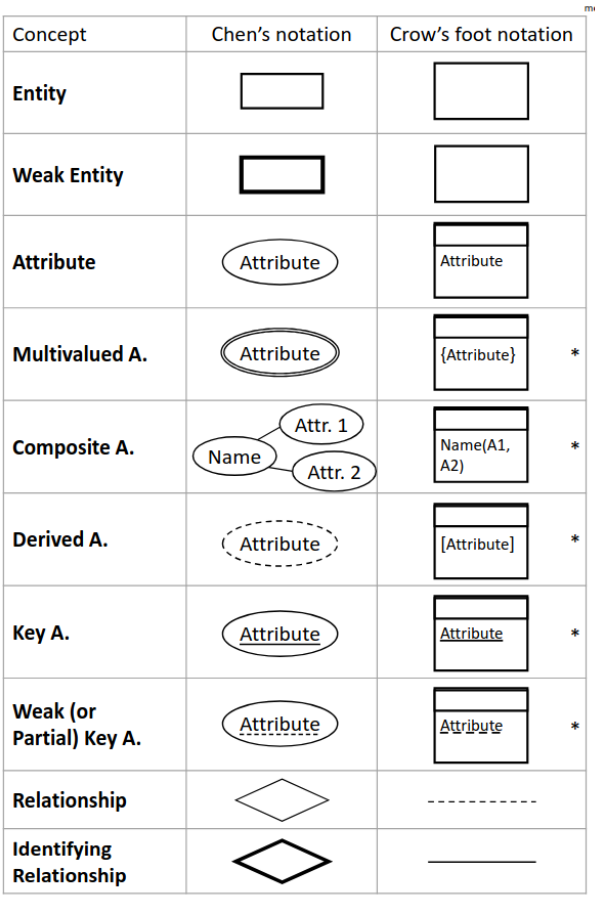
TIMESTAMP(fsp) A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC.

TIME(fsp) A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59'

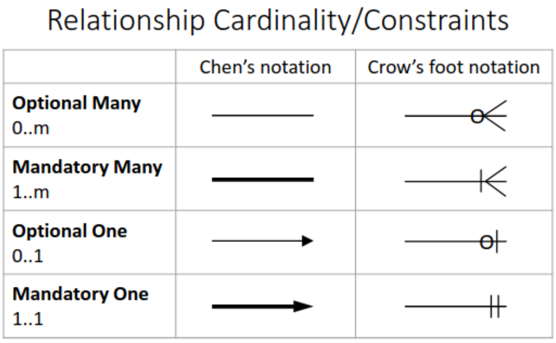
YEAR A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000.

Data dictionary is metadata describing the schema of the table produce in database design. It contains: keys, attributes, data types, null, uniques, and description.

Chen diagram, Crows-Foot diagram

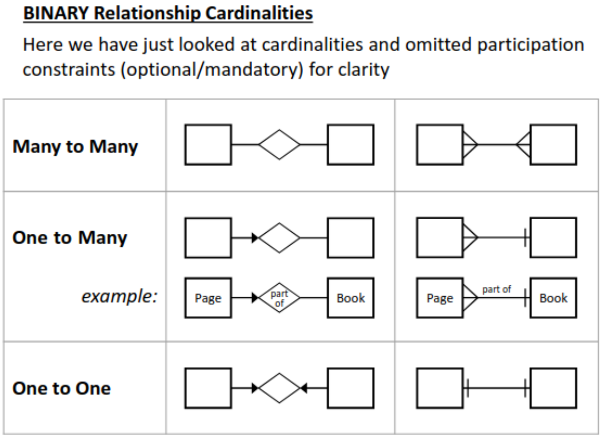


Concept	Chen's notation	Crow's foot notation
Entity		
Weak Entity		
Attribute	Attribute	Attribute
Multivalued A.	Attribute	{Attribute} *
Composite A.	Name Attr. 1 Attr. 2	Name(A1, A2) *
Derived A.	Attribute	[Attribute] *
Key A.	Attribute	Attribute *
Weak (or Partial) Key A.	Attribute	Attribute *
Relationship		
Identifying Relationship		



Relationship Cardinality/Constraints

	Chen's notation	Crow's foot notation
Optional Many 0..m		
Mandatory Many 1..m		
Optional One 0..1		
Mandatory One 1..1		



BINARY Relationship Cardinalities

Here we have just looked at cardinalities and omitted participation constraints (optional/mandatory) for clarity

Many to Many		
One to Many *example:*	Page part of Book	Page part of Book
One to One		

3 Data Modeling

Definitions

Entity A real world object distinguishable from other objects. Defined by a unique set of attributes.

Entity Set A list of entities of the same type.

Relationship Entity has a key for association in a relationship between two or more entities. Relationship can have attributes

Relationship Set A collection of relationships with same type

Key A minimal set of attributes to identify an entity in a set. A form of integrity constraint. A key is a superkey that is a minimal set (no subset is a superkey)

Superkey A set of fields where no two distinct entities have the same set.

Primary, Candidate keys One key is chosen to be primary, and all other keys are candidate keys

Foreign keys a set of fields in one relation referring to a tuple (via primary key) of another relation.

Constraints

Integrity Constraint Constraints on data in a relation, includes PK constraints, FK/referential constraints, domain constraints

Domain constraints Domain specific restrictions on attributes

Primary key constraints Entities in an entity set are all unique, via primary key

Foreign Key constraints PK and FK to ensure validity of relationships between entities (cannot point to non-existent). If all key constraints are satisfied, there is referential integrity.

Key Constraints the maximum number of entities in a relationship. many-to-many, one-to-many, one-to-one

Participation Constraint The minimum number of relationships an entity must participate in. Total participation means at least one, partial participation means at least zero.

Legal instance A relation instance that satisfies all ICs. DBMS should not allow illegal instances

Enforcing constraints. Under FK relationship

- Reject adding tuple with non-existent FK
- Deleting/Updating PK need to: cascade deletion with associated FK; disallowed if any PK dependents; set FK to null

Weak entities

- Identified uniquely by another owner/strong entity/entities through an identifying relationship (set).
- The identifying relationship must have the weak entity take total participation in one-to-one, or one-to-many (uncommon).
- Only have partial keys, uniquely identified only by its partial key combined with its owner's primary key (but has an identifier/primary key).
- Created by business rules.

Types of relationships due to key/participation constraints on one entity

Optional-many 0..m, any number of relationships

Mandatory-many 1..m, one or more relationships

Optional-one 0..1, at most one relationship

Mandatory-one 1..1, must participate one relationship (weak entity)

Special attributes

Multi-valued attributes contains multiple values of the same type (array, homogeneous)

Composite attributes have a structure inside with elements of different types (struct, heterogeneous).

Crow's foot terms

Identifier A key

Partial identifier Partial key

Mandatory attributes NOT NULL, blue diamond

Optional attributes NULL, empty diamond

Derived attributes square brackets

Multi-valued curly braces, multiple homogeneous types

Composite attributes parenthesis, multiple heterogeneous types

Relationship degree number of entities sets participating in a relationship, unary, binary, ternary

Relationship cardinalities mapping cardinalities, cardinality ratios, is the number of other entities that can be associated with an entity in an entity set through a relationship (depends on key constraints, many-to-one implies one entity set has m cardinality and one entity set has 0..1 cardinality)

Note that Crow's foot flips the cardinality orders compared to chen's notation. In Chen, between A and B, the number of relationships that entities in A can participate in is on A's side, while this is flipped for crow's. Omitting participation constraints on Crow's defaults to partial participation.

Conceptual design

- Model entities, attributes, relationships.
- Captures semantics of data via relationships (address as attributes or entities)
- Follows requirement analysis, yields high level description of data stored, subjective

Relational model

Data model Translates real world objects into structured data (data model). Includes relational, OO, network, hierarchical

Entity sets are represented by rows (tuples) and columns (fields)

Relationships are represented by PK and FK.

Relational database is a database using the relational model. It contains relation sets.

Relation is made of a schema and an instance. Commonly, relation is the relational instance

Relation Schema specifies the relation metadata like name, column/attribute names and types

Relation Instance is a table with rows and columns. Cardinality is number of rows, degree/arity is number of columns. All rows are distinct with no ordering

A relation schema is written like `Employee(ssn: string)` with the type being optional.

Conceptual to relational

- entity sets become relations, entity attributes becomes relation attributes
- multi-valued attributes are unpacked into entities or lookup table, so are composite attributes into attributes
- many-to-many relationships require a new associative entity set (relation) with the FKs of participating entity sets forming a composite PK, and also descriptive attributes. The associative entity is weak
- one-to-many relationships adds a FK of the relation with key constraint (one-side). If total participation, put NOT NULL on the FK, can put ON DELETE NO ACTION or ON DELETE RESTRICT on FK.
- one-to-one places the FK depending on business rules. The PK on the mandatory side is a FK on the optional side (place it where it is least likely to be NULL). If both are optional or mandatory, arbitrarily pick one. Include descriptive attributes on the FK side
- weak entities combines the weak entity set and identifying relationship set into the same relation like one-to-many. Must specify ON DELETE CASCADE on FK, so that owner deletion implies all weak entity deletion

For Unary relationships, do the transition on the same entity set. For ternary many-to-many, create three way weak associative entity.

Database design process summary

conceptual design create ER diagrams of entities and relationships

logical design create relational data models

physical design choose the data types and nullity of the relational fields

implementation SQL DDL statements

4 Relational Algebra

Theory behind SQL, ensures it produces the correct answers.

Queries is a sequence of operations, each with inputs and outputs as relations

Selection horizontal filtering, selects a subset of rows. $\sigma_{a\theta b}(R)$ selects on R, identical schema, can't have duplicates.

Projection vertical filtering, retains a subset of columns. $\pi_{a_1,a_2,\ldots}(R)$ retains attributes on the projection list in R, with the same name. In RA, this removes duplicates. SQL doesn't

Union combines tuples in one with tuples in another, schema must equal (Union compatible). $R \cup S$ combines R and S, must have same number of fields with same types. Duplicates will be removed

Set difference , filter/include tuples in one that is not in another, union compatible. $R - S$ retains rows in R but not in S.

Cross product , combine two relations tuple by tuple and combined schema. $R \times S$ merges the rows with attributes inherited both R and S, may need to rename.

Renaming $\rho(C(a \to b), R)$ renames a to b in R into relation C. Also can use field index from 1 in place of attribute names

The conditions uses the typical operators where AND and OR are $\wedge$ and $\vee$.

Compound operators are combinations of basic operators with no added computaitonal power

Intersection Retains rows appearing in both relations, union compatible. $R \cap S = R - (R - S)$.

Natural Join $R \bowtie S$ is a cross product that selects rows where equal named attributes in both relations are equal (keeping only one version in final schema)

Conditional Join theta join, cross product with condition, $R \bowtie_\theta S = \sigma_\theta(R \times S)$

Equi-Join a conditional join where condition is equality. different to natural join for we keep the duplicated columns and can also arbitrarily join

5 SQL

SQL supports CRUD: create, read, update, delete. Its commands include

- DDL, data definition language to define the db
- DML, data manipulation language to maintain and use the db
- DCL, data control language to control access to the db
- Other commands for admin and transactions

Always use single quotes. Commands, attributes are case insensitive. Table names depend on OS.

DDL create table

```
CREATE TABLE Customer(
    customer_id SMALLINT AUTO_INCREMENT,
    name VARCHAR(100)
    PRIMARY KEY (customer_id)
);

CREATE TABLE Account(
    ...
    customer_id SMALLINT NOT NULL,
    FOREIGN KEY (customer_id)
        REFERENCES Customer(customer_id)
        ON DELETE RESTRICT
        ON UPDATE CASCADE
);
```

Actions are

NO ACTION, RESTRICT Default, restricts deletion (or changing PK) of PK record and throws error if did

CASCADE Delete FK row on deletion of PK row. Changing PK changes FK.

SET NULL Set FK to NULL when PK delete (or on update)

SET DEFAULT Set FK to Default Value (or on update)

DDL views

- Virtual relations not in the physical design, but available to users
- Hide query complexities, hide data from unauthorized users
- Improves data security by granting different views on same relation for different users
- In SQL, can be used just like any other table.

```
CREATE VIEW name AS
SELECT ...;
```

DDL alter drops or inserts attributes

```
ALTER TABLE table ADD
field1 type1;
ALTER TABLE table DROP
field1;
ALTER TABLE table RENAME COLUMN old TO new;
```

DDL rename renames tables

```
RENAME table old TO new;
```

DDL truncate removes everything from table, faster than DELETE but cannot be rolled back

```
TRUNCATE table;
```

DDL drop kills a relation with its schema. No undos

```
DROP TABLE table;
```

DML insert table

```
INSERT INTO Customer
    (first_name, last_name)
VALUES ('Peter', 'Smith');

INSERT INTO Customer
VALUES (DEFAULT, 'Peter')
       (DEFAULT, 'Terry');

INSERT INTO table
    (SELECT * from table2);
```

- Default implies the default value (auto increment or null).
- Inserting using columns implies the rest are default.
- Query schema must match table schema

DML replace is identical to insert except that it replaces rows inserted that match records with the same PK. Insert will throw an exception here.

```
REPLACE INTO table
VALUES (values);
```

DML update. Defaults to update all rows. Would trigger key constraint checks.

```
UPDATE table
SET field1 = field1 * 1.1, field2 = field2 * 1.2
WHERE ...;

-- also CASE as an multi-branch for a column/value
UPDATE salaries
SET salary = CASE
WHEN cond1 THEN value1
WHEN cond2 THEN value2
ELSE value3
END;
```

DML delete. Default to all records. Would trigger key constraint checks.

```
DELETE table
WHERE ...;
```

DML select. Square brackets are optional, curly are choices.

```
SELECT [ALL|DISTINCT]
    {columns}
FROM table1, table2
WHERE ...
GROUP BY {column|expr} [ASC|DESC]
HAVING ...
ORDER BY {column|expr} [ASC|DESC]
LIMIT { [offset] count | count OFFSET offset }
```

Important SELECT features

- Can do maths like `SELECT 1+2`.
- orders of expressions in select matters
- Default ordering is ASC.
- DISTINCT implies unique rows, ALL is default
- FROM with multiple tables implies cross product
- WHERE only works outside of GROUP BY
- Logical keywords are AND, OR, NOT. And basic comparison operators (<> is not equals).
- LIKE is also a logical operator with `column LIKE pattern`, and pattern is a string with wildcards % for any number of characters, and _ for single character
- Rename expressions in the columns (and tables) using AS.
- Disambiguate fields with the same name by `table.field`. Select all fields from table using `table.*`.

Aggregation function operate on the subset (or entire) of values in a column and returns a single value: AVG, COUNT, MIN, MAX, SUM. Importantly

- All except `COUNT(*)` ignores NULL and returns NULL if all NULL. `COUNT(*)` counts all records including NULL, while `COUNT(col)` only includes non-null.
- Also has `COUNT(DISTINCT col)`.
- Using aggregate functions without groupby will result in one row of aggregated results

DML joins.

```
-- Cross product
SELECT * FROM R, S;

-- Equi-join
SELECT *
    FROM R INNER JOIN S
    ON R.id = S.id;

-- Natural join
-- Cross product if no same-named attributes
SELECT *
    FROM R NATURAL JOIN S;

-- OUTER JOINS
SELECT *
    FROM R LEFT OUTER JOIN S
    ON R.id = S.id;
SELECT *
    FROM R RIGHT OUTER JOIN S
    ON R.id = S.id;
```

- Inner joins include records only if the condition matches.
- Outer joins includes matching records, but also non-matching records on one side filled with NULL in empty columns.
- Full outer join doesn't exist

Useful functions

```
UPPER(str), LOWER(str), LEFT(str, n), RIGHT(str, n)
COALESCE(value, default), DATE("2024-10-22")
```

```
IF(cond, a, b)
```

DML set operations. UNION removes duplicated, UNION ALL to keep

```
SELECT ...
UNION
SELECT ...;
```

DML subqueries are nested select statements that produces a relation (single column is table set). They can be used to perform set tests or set comparisons like

```
SELECT ...
WHERE
id IN (...)
id NOT IN (...)
id > ANY (...)
id > ALL (...)
EXISTS (...);
```

We can also explicitly write sets in parenthesis `(10,20,30)`. Note that subqueries has a performance penalty.

DCL admin commands

CREATE USER

DROP USER

GRANT for assigning privileges and roles for users and roles

REVOKE for revoking privileges

SET PASSWORD

ANALYZE TABLE table for table statistics

CHECK TABLE table checks for errors

DESCRIBE table to obtain table schema

EXPLAIN expression for query execution plans

SHOW DATABASES and USE database to show and choose a specific db

6 Storage

Components of DBMS

- Query Processing Modules
- Storage module
- Concurrency control module
- Crash recovery module
- Database files

Data is stored on disk, we need to swap them to memory to process them. Swapping data in and out of RAM is called paging.

Storage hierarchy for speed, cost, size tradeoffs

- CPU registers
- Cache
- RAM
- Hard disk
- Tape

Storage terms

- Database contains files
- Files is a collection of pages
- Page is a collection of records

File organization methods

Heap files : no particular order between records, implemented as a doubly linked list of pages, suitable for full scan accesses. fast inserts due to fast disk page allocation

Sorted files : pages and records ordered by some condition, implemented as a doubly linked list of pages, but sorted according to some columns, best for ranged ordered retrieval of records, slow inserts due to reshuffling

Index files : contains indexes along-side files that speed up retrievals in some order

Assign a cost for all operations as the number of page accesses (or IOs). Decide file organization method by typical operation IOs.

Indexes

- As index files, containing data structures on top of data pages used to speed up searches
- Built over specific fields called search key fields (any subset of fields in relation, non-unique), only speed up searches on these search key fields
- Contains data entries. The entries are sorted by search keys and contains the search keys and pointers to data pages containing the records
- B+ tree has directories, which are inside nodes that has no pointers to data records but contains partitioning information. Data entries are always leaf nodes.

Classification of indexes

Clustered Clustered indexes have the data entries and data records in the same order. Otherwise it is unclustered. A data file can have at most one clustered index on one particular search key. Clustered implies sorted files. For clustered, retrievals and range searches are cheaper, but harder to maintain on inserts.

Primary Primary indexes has the search key on the table's primary key. Secondary indexes doesn't. Primary indexes have no duplicated search keys

Composite Single key index has a single column search key. Composite key index has a subset as a search key.

Technique The underlying data structure. Hash-based use a hash table to locate the data entries. It maps the search keys to the bucket containing data entries. Only good for equality selections. Tree-based use a B+ tree. Contains directories with boundary search key values and pointers to lower levels, leafs containing data entries. Good for range selection and equality searches.

In practice B+ tree with heap file is more efficient than sorted files.

7 Query Processing

Shortcuts when performing operations (benefits of relational db)

Evaluation Clever operation implementations

Optimzer Exploiting equivalencies of relational operators

Cost model To choose between alternatives

But there are no universally superior implementations/techniques for operations.

Query processing/optimization workflow

- Break query into blocks of SELECT statements.
- Convert each block to RA
- Use catalog manager for table schemas and statistics
- For each block, optimize the query by continuously generating execution plans through RA equivalences, then estimating costs
- Outputs a query plan, evaluated by the executor

Terms

Query execution plan The sequence in which tables are accessed, the methods used to extract data from table, and methods used to compute calculations, including filtering, aggregating/joining, and sorting data.

Executor takes a query execution plan and actually runs it

Access path is the way that records in a table is fetched (heap, index, sorted files)

Reduction Factor selectivity, estimates the proportion of relation that will qualify for a predicate, estimated from catalog information. Combination of RF uses multiplication.

Projection Factor proportion of attributes keeping against all attributes, similar to RF

Main memory Let main memory buffer be pages to store temporary data. Let B be the number of buffer units.

Catalog Contains schema information and column statistics, updated periodically

Query block a SELECT statement. These are units of optimization and we begin by optimizing the inner most block first.

Catalog contains

- relation and index information, and column statistics item
- Number of tuples and pages per relation
- distinct key values for each column
- low and high key values for each attribute
- index height and pages
- reduction factors

Reduction factors

Col=Value Assume uniform distribution $1/NKeys(col)$

Col>Value $(High(col) - val)/(High(col) - Low(col))$

Col<Value Similar using $Low(col)$ instead

a<Col<b Use $(b - a)/(High(col) - Low(col))$

Col1=Col2 Joins , use $1/\max(NKeys(col1), NKeys(col2))$

No information use magic number 1/10

Result size can be approximated by the RF and relation pages

$$\text{NPages}(result) = \prod_j \text{NPages}(R_j) \prod_i RF_i$$

Processing selections (access path), costs and process

No index unsorted Heap scan $\text{NPages}(R)$

No index sorted B-search $\log(\text{NPages}(R)) + \text{NPages}(R)RF_i$

Single Tuple For b+tree $Height(I) + 1$, for hash 2.2

Indexed clustered Need to find data entries, then data records. Data entries have $\text{NPages}(I)$ pages, and are always sorted. B+tree $(\text{NPages}(I) + \text{NPages}(R))RF_i$, hash $2.2\text{NPages}(R)RF_i$

Indexed unclustered B+tree $(\text{NPages}(I) + NTuples(R))RF_i$, hash $2.2NTuples(R)RF_i$

B+tree usage B+tree index can match a combination of predicates using attributes in the prefix of its search key. The prefix predicates are called matching predicates or primary conjuncts; the RF will only include RF from these matching predicates

Hash tree Requires predicates that includes all search keys. While B+tree can process all predicate types, hash can only match equality predicates

1 Find cheapest access path by estimating IOs

2 Retrieving tuples using access path, using matching predicates

3 Apply non-matching predicates to discard some retrieved tuples. These selections on the non-matching predicates after the access path are said to be done "on-the-fly".

Processing projections, mainly removing duplicates

Sorting Scan and retrieve relevant fields, sort the result, remove adjacent duplicates. Use external merge sort with P passes

$$ReadTable + WriteProjected + Sort + ReadProjected$$
$$\text{NPages}(R) + \text{NPages}(R)PF_i + 2P\text{NPages}(R)PF_i + \text{NPages}(R)PF_i$$

Hashing Scan relation extract needed attributes, hash records into buckets, remove duplicates within each bucket. Use external hashing where pointers to partitions are stored in the $B - 1$ buckets. Hash function is constant time so no sorting step

$$ReadTable + WriteProjectedHash + ReadProjected$$
$$\text{NPages}(R) + \text{NPages}(R)PF_i + \text{NPages}(R)PF_i$$

Index scan If we have an index with search keys as projection fields, we can page only the data entries. This reduces the cost of ReadTable in both sort and hash projections to $\text{NPages}(R)PF_i = \text{NPages}(I)$.

Processing joins, focus on conditional joins on one column.

Definition Left relation is outer relation, right relation is inner relation

Conditions Nested loop joins works for all conditional joins. Sort-merge and hash join only works for equality joins.

Simple nested loop join For each record in outer, scan entire inner relation to match.

$$\text{NPages}(Outer) + NTuples(Outer)\text{NPages}(Inner)$$

Page oriented nested loop join For each page in outer, scan entire inner relation to match.

$$\text{NPages}(Outer) + \text{NPages}(Outer)\text{NPages}(Inner)$$

Block nested loop join Assigns one memory buffer for inner relation, one for the output, and $B - 2$ for block storing outer relation. Read next $B-2$ outer relation pages into block, scan through entire inner relation, storing matching pages, repeat.

$$\text{NPages}(Outer) + NBlocks(Outer)\text{NPages}(Inner)$$

where $NBlocks(Outer) = ceil(\text{NPages}(Outer)/(B - 2))$.

Sort Merge Join Sort outer and inner relation on join column, two pointer merge on sorted relation.

$$2P\text{NPages}(O) + 2P\text{NPages}(I) + \text{NPages}(O) + \text{NPages}(I)$$

due to external merge sort with P passes. Useful if relations are already sorted due to access path, or if output is required to be sorted on the join column.

Hash join Partition both relation into hash buckets. Then reading matching pairs in each bucket.

$$2\text{NPages}(O) + 2\text{NPages}(I) + \text{NPages}(O) + \text{NPages}(I)$$

8 Query Optimization

Reasons for query optimization

- Many equivalent ways of executing the query
- Cost varies significantly between alternative plans
- Want to find the query plan with lowest cost

Details of query plan

- A tree with RA operations as inner nodes and access paths as leaves
- Each operator and access path is labeled with a choice of algorithm (access path, join, on-the-fly algorithms)
- Total cost of query plan is the sum of the operator/access-path costs

Details about operator equivalence

- Changes in operators that don't change the results. Used by query optimizers to generate query plans
- Selection operator has cascade and commute equivalences

$$\sigma_{c_1 \wedge c_2}(R) = \sigma_{c_1}(\sigma_{c_2}(R))$$
$$\sigma_{c_1}(\sigma_{c_2}(R)) = \sigma_{c_2}(\sigma_{c_1}(R))$$

- Projection operator has cascade equivalence, where $a_1 \supset a_2$

$$\pi_{a_1}(R) = \pi_{a_1}(\pi_{a_2}(R))$$

- Joins are associative and commutative

$$R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T$$
$$R \bowtie S = S \bowtie R$$

- Mixing joins, cross products, and selection projections

$$\sigma_{s_1=r_1}(S \times R) = S \bowtie_{s_1=r_1} R$$
$$\sigma_{s_1}(S \bowtie R) = (\sigma_{s_1}(S) \bowtie R)$$
$$\pi_{s_2}(S \bowtie_{s_1=r_1} R) = \pi_{s_2}(\pi_{s_1,s_2}(S) \bowtie_{s_1=r_1} \pi_{r_1}(R))$$

Consider the result size of a query block in the number of tuples. All predicate RF are included.

Single table $NTuples(R)\prod_i RF_i$

Cross product $\prod_j NTuples(R_j)$

Joins Note that we treat conditional joins as cross products with predicates $\prod_j NTuples(R_j)\prod_i RF_i$

The types of query plans are: single relation query plans and multiple relation plans. The main costs are in the access path and joining, with non-matching predicates performed on-the-fly with no additional costs.

Single relation query plan

- Consider each access path (heap, index, sorted). Always consider heap scan.
- Other non-matching predicates or project is done on the fly.
- Result size may depend on access path if selection is done .
- Using index but no matching predicates implies $RF = 1$

Multi-relation query plan

- Steps are: select relation join orders, select join algorithm for each join, select access method for each relation
- The number of query plans is $N!J^{(N-1)}A^N$ where N is the number of relations, J is the number of join algorithms, and A is the number of indexes
- Limit to left-deep join trees, for they have fully pipelined plans where intermediate results are not written to temporary files but are directly used in the next calculations
- Prune out cross product trees and only use conditional joins

Key points in cost computation

- When joining against an intermediary relation, estimate its relation size, and apply joining algorithm using the estimated size. Assume that it is already read and subtract one NPages(R) cost from the join
- After SMJ, the relation is already sorted on the index
- A clustered index access path is sorted on the search key for SMJ
- Heap scan is never sorted

9 Normalization

Terms

Universal relation is a table containing every relation in denormalized form

Normalization Separation of records by attributes into their own tables. A technique that removes unwanted redundancies.

Denormalization Storing records all in a large table with many columns. Contains duplicates and potential inconsistencies

Denormalization anomalies

Insertion Cannot add a new value into a column without at least one record spanning all fields

Deletion Removing one record may lose an entirely unique value in one field

Update Changing one attribute for a value will change multiple records, or inconsistency

Normalization benefits

- Minimum redundancy, no inconsistency, no anomalies, no performance issues

Denormalization Benefits

- Faster queries, for joining is slow
- Improve performance on time critical operations

Functional Dependency

- Dependency between fields (sets of fields) in a relation
- Attribute X determines Y if each value of X uniquely identifies with one value of Y across all records. Then $X \to Y$.
- X determines Y, X implies Y, Y dependent on X, Y upon X.
- Determinants are attributes on the LHS of the arrow
- Key attributes are attributes that are subsets of the primary key (only on PK), non-key attributes are not key attributes
- Partial functional dependency is a functional dependency of non-key attributes on a subset of the PK but not all.
- Transitive dependency is a functional dependency between two non-key attributes.

Armstrong's axioms

Reflexivity $B \subseteq A \implies A \to B$

Augmentation $A \to B \implies AC \to BC$

Transitivity $A \to B \land B \to C \implies A \to C$

Normalization forms

Inheritence Normal forms are inherited, higher order implies lower order

First NF keep atomic data and remove repeating groups. Let repeating groups as attributes with multiple values per record (not representable in relational models with 2d tables), and atomic data as attributes with singular values. Need to convert repeating groups into a new relation with FK linking back.

Second NF Remove partial dependencies. Take both sides of partial dependency into new relation with PK subset as composite PK, add PK/FK in original relation linking to composite PK in new relation. Can have anomalies between determinant and non-key attributes.

Third NF Remove transitive dependencies. Extracting attributes into new table with the determinant as PK, then put FK in original record. Can have anomalies between determinant and dependent.

Boyce Codd NF Every determinant is a candidate key (or superkey). Relations not in BCNF but are in 3NF have functional dependencies of key attributes on non-key attributes, or key attributes on key attributes. Separate non-candidate key determinant and dependent into new relation with determinant as PK, add PK/FK in the original relation. Can have anomalies.

Normalization cannot fix all design flaws, like designs where an attribute is represented by the specific table instead of as a column.

10 Transactions

Terms

Transaction A logical unit of work (atomic unit of work). Contains a sequence of DML or SQL statements. It solves the defining units of work and concurrency problem.

Atomic unit A unit of work that must be either entirely completed or aborted

Consistent state All IC in the DB are satisfied

Atomic units

- DML or SQL statements are atomic by themselves
- Create user-defined atomic units of work via transactions.
- Successful transaction changes the db from a consistent state to another consistent state

ACID property of transactions

Atomicity Transactions are treated as a single, indivisible, logical unit of work. All operations must either be completed or aborted with everything undone.

Consistency Constraints that held before a transaction must also hold after it. Multiple users accessing the same data must see the same value

Isolation Changes made during the execution of a transaction cannot be seen by other transactions until this one is finished

Durability Completed transactions makes permanent changes to the db, even if db fails after.

Defining units of work

- Single SQL statements are transactions. If db crashes in the middle, the transaction is reversed and no records will be changed after restart.
- Multiple statements can be combined into a user-created transaction.
- Business rules may require transactions. Features of units of work: series of statement embedded in a larger application, indivisible units, rollback on errors keeping database consistency
- Errors during a transaction will reverse all SQL statements, with the user being able to retry it later.

SQL Transaction

```
-- implicit after commit/rollback
START TRANSACTION;

-- SQL statements...

-- explicit commit
COMMIT;  -- or ROLLBACK;
```

Concurrent access is the concurrent execution of DML on shared db by multiple users. Its problems are

Lost updates Transaction that writes last overwriting the writes of the first transaction.

Uncommitted data First transaction modifies data then rollback. Second transaction accesses the uncommitted data and writes result based on that

Inconsistent retrievals When aggregating over a db, another transaction updates the data, so some aggregating data is read before and some after the change.

Solutions to concurrency

- A schedule is the order that transactions runs in. It is serializable if multiple concurrent transactions appear as if they were executed one after another
- Serializable schedules only need to appear as running it one by one, just need to have the exact results. True serial execution with no concurrency is expensive.
- Transactions should run in a serializable schedules. This ensures consistent results without problems.
- In reality, DBMS creates a schedule of reads and writes for all concurrent transactions, then interleaves their execution using concurrency control algorithms.

Concurrency control algorithms are: locking, timestamp, optimistic concurrency control

Lock Grants exclusive use of data to a transaction. Need to acquire lock prior to data access, then release it after. Another transaction needs to wait to acquire the lock on the same data. Main bottleneck is the waiting time to acquire a lock

Availability Measures how likely we need to wait to access the data

Lock manager Responsible for assigning and controlling locks used by transactions

Database level Locking entire db. Good for batch processing (large infrequent) but bad for multi-user DBMS.

Table level Cause bottlenecks if two transactions want to access different parts of a table, not good for highly multi-user db

Page level

Row level Improves data availability, high overhead in storing locks and using locks. Most popular

Field level Most flexibility with highest data availability, extremely high overhead, not common.

Binary locks Two states for both read and write. Eliminates lost update problem. Too restrictive for optimal concurrency for locks on reads.

Exclusive locks Reserves read/write access. Must be obtained to write. Granted if no other locks are held

Shared locks Grants read access, must be obtained to read, acquired if no exclusive locks.

Deadlock Two transactions wait for each other to unlock data. Only happen with exclusive and binary locks. Dealt but prevention (graphs) or detection (timer and timeouts). MySQL automatically detects deadlocks and rollbacks the transaction that detects it

Timestamp Assign unique timestamp for each transaction. Each data item accessed by a transaction gets its timestamp. When transaction wants to read or write, compare current transaction timestamp against attached timestamp and allow if current time is later, rollback otherwise.

Optimistic Executes transaction without restrictions. When committing, checks for data its read for alteration, rollback if so. Assumes that a majority of database operations do not conflict.

Logs

- In general contains updates to data and transaction statements
- A log for the beginning of transaction and one for the end
- For each SQL statement, log contains operation performed, objects affected, before and after values, points to previous and next transaction logs
- Grants ability to restore database to previous consistent state when aborted. Or to restore a crashed database to a checkpoint.
- A checkpoint is a time when all committed data are consistent, all uncommitted transactions are recorded, and all logs are kept. If system failure, restore to previous checkpoint and examine the logs for uncommitted transactions and restart.

11 Capacity Planning

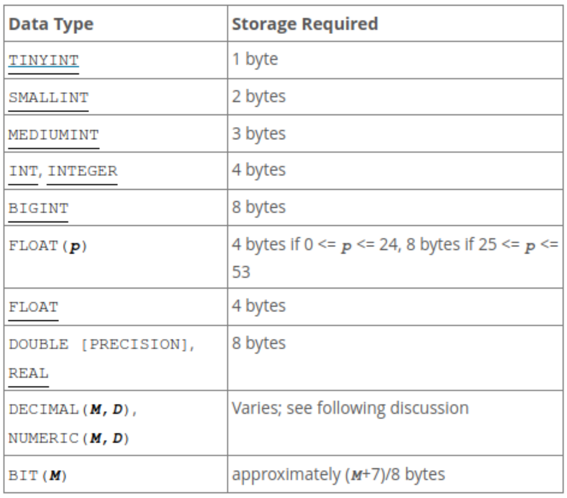
Capacity planning considers

Disk space Predicting future data size that may saturate the system, and delaying it. Depends on relation data volumes. Done in system design

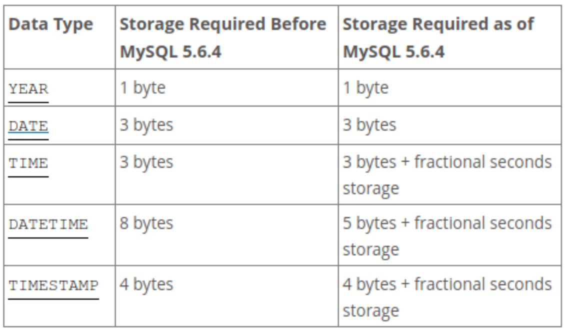
Transaction throughput Depends on relation access frequencies. Done in system design

Continuous maintenace Monitor and predict disk usage and transaction load between go-live and throughout the system life. Done in system maintenance

Data type sizes



Data Type	Storage Required
TINYINT	1 byte
SMALLINT	2 bytes
MEDIUMINT	3 bytes
INT, INTEGER	4 bytes
BIGINT	8 bytes
FLOAT(*p*)	4 bytes if 0 <= *p* <= 24, 8 bytes if 25 <= *p* <= 53
FLOAT	4 bytes
DOUBLE [PRECISION], REAL	8 bytes
DECIMAL(*M*,*D*), NUMERIC(*M*,*D*)	Varies; see following discussion
BIT(*M*)	approximately (*M*+7)/8 bytes



Data Type	Storage Required Before MySQL 5.6.4	Storage Required as of MySQL 5.6.4
YEAR	1 byte	1 byte
DATE	3 bytes	3 bytes
TIME	3 bytes	3 bytes + fractional seconds storage
DATETIME	8 bytes	5 bytes + fractional seconds storage
TIMESTAMP	4 bytes	4 bytes + fractional seconds storage

Estimating disk space requirements, an estimation as DBMS stores more than just records

Database size The sum of the table sizes in bytes.

Table size The product of cardinality and average row width

Row width Sum of storage size of fields, depends on data type. For numeric or datetime, use cheatsheet. For varchar or blob, use average size in bytes from catalog.

Growth forecast Gather estimates of rate of new records from system analysis business rules, multiply by row size to estimate disk size growth rates

Estimating transaction load

- Multiply frequency of transaction by the number of SQL statements per transaction
- Limit the processing time of a transaction to ensure performance

12 Backups and Recovery

A backup is a copy of the database. It aims to restore the data if the data is lost. A backup strategy details how data is backed up and what is the recovery process.

Type of errors (failure causes)

Human errors Accidental drop or delete (most common)

Hardware or software Malfunction like bugs, hard drive failure, corruption, CPU memory failed

Malicious activity security compromise on servers, db, applications

Natural or man-made disasters

Regulation archiving rules, metadata collection, privacy rules

Types of failures are

Statement SQL syntax issue

User process Database process fails

Network Between user and db

User error Accidental drops

Memory Memory is corrupt

Media and disk corruption

Backup categories

Physical backup Stores exact copies of files, logs, and directories. Suitable for large databases requiring fast recovery. Offline backup and recovery (can use locks to do it online). Only portable to machines with similar configuration

Logical backup Use SQL queries to store relations into files. Slower than physical in backup and recovery. Output larger than physical for no compression. No log or config files. Machine independent. Online backups. Use `SELECT ... INTO OUTFILE file` and `LOAD DATA INFILE infile`.

Hot backup Online backup, backup occurs when database is live. Clients don't realize a backup is happening. Requires locking to ensure data consistency.

Cold backup Offline backup, backup occurs when db is stopped. To maximize availability, take backup from replication server. Simpler for no locks. Preferable method but not possible in all situations (especially if no downtimes).

Full backup Backups the entire database, includes everything needed to get the db operation in an entire db failure

Incremental backup Stores changes since the last backup, only the logs. To restore, stop db, restore using full backup, copy incremental log files to disk, start db and ask it to redo the logs.

Offsite backups Backups stored physically not near the server (tapes in vaults, remote replicate mirror db, cloud backup). Enables disaster recovery

Onsite backup Backups stored in the same server as db

Recovery process

- Use backup to recover db to last backup state
- Use crash recovery transaction logs to recover to last checkpoint to recover all committed transactions and uncommitted transaction states

13 Data Warehouse

OLTP vs OLAP

Managers wants to know statistics/metrics about company. Use data encoded in databases for analysis and business decisions

OLTP Transactional database. Relational db are Online Transaction Processing DB (OLTP) suitable for processing day-to-day operations, is efficient and automates business processes

OLAP Informational database. Integrated way to get entire organization's data, a single db that stores all data in a form that supports business decision-making.

Transactional queries Narrow with simple queries on few tables

Analytical queries Complete queries that are board, accessing multiple tables. Contains a numerical aggregation part (sales) and a dimension part (by store).

Problems of OLTP Not designed to process large, complex, aggregation queries

Too many databases with different types of DBMS

Problems of duplicated data, inaccessible data, inconsistent data when scaled

Comparison between transaction and informational db

Feature	Transactional	Informational
Primary Purpose	Run day to day buisness	Support decision making
Type of data	Current data representing current state of business	Historical data, snapshots and potentially predictions
Primary Users	Customers, Clerks, other employees	Managers, analysts
Usage Scope	Narrow, planned, fixed interfaces	Broad, ad hoc, complex interface
Design Goal	Performance, availability	Flexible and data accessibility
Volume	Constant updates and queries on a few rows and tables	Periodic batch updates, complex query on multiple tables or rows

Data warehouse is an informational database

- Single repository of organizational data
- Integrates data from multiple sources, automatic extraction, transformation, and loading into warehouse
- Makes entire data available to managers
- Supports analysis and business decision-making
- Read-only, large data store with several Tb or Pb of data
- Supports analytical queries without reducing operational db performance

Advanced data warehouse features

Subject oriented Organized around particular subjects

Validated and integrated data Combining data from different systems into a common format for comparison and consolidation. Validation of data sources

Time variant Contains historical data organized as snapshots with time stamps. Allows trend analysis

Non-volatile Only read access and updates done automatically by extract transform load process by database admin periodically

Architecture of DW

- Extract data from data systems, internal or external sources
- Process data in staging area. Cleaning, matching, deduplication, standardization
- Load data into storage area. Contains metadata, data warehouse, smaller data mart by subject
- Data from DW fed into analytic and reporting tools. Dashboards, adhoc queries, modeling, visualization, applications,

data mining. (Business intelligent dashboard is a software that provides information about the financial state of the business through visualizations)

Dimensional modeling

Dimesional Analysis Business analyst analyze by focusing on single indicator, then going in dimensions to solve a problem. Has a single fact and a series of dimensions.

Dimensional modeling A database model that supports dimensional analysis queries stored in DW

Star Schema A dimensional model based on the multi-dimensional model of data, designed for retrieval only databases. Preferable over OLTP for they are organized around facts and dimensions that resembles dimensional analysis, also denormalized so faster in aggregating and querying data.

Dimensional model is a restricted ER model, containing

- Fact tables
- Dimensional tables associated with the facts
- Hierarchies in the dimensional tables

Steps of dimensional model design

- Choose business process (operation)
- Choose measured facts
- Choose granularity of fact table
- Choose dimensions
- Complete dimension tables with hierarchies

Terms

Facts Business aggregated metrics, stored as non-key attributes in fact table.

Dimension A factor that we can classify a fact by. Stored as PFK in fact tables pointing to dimension tables.

Hierarchies For dimension, starts specific then goes general. Embedded as attributes of dimension table or separated into new dimension tables. One-to-many relationships from general to specific tables.

Hierarchy normalization The choice of hierarchies is a choice of normalization. If normalized, eliminates redundancy, creates storage efficiency, referential integrity. If denormalized, faster querying, useful for analysis. Usually want denormalized tables in DW for performance and analysis.

Granularity Level of detail for the facts. Depends on keys used, the finest detail of a fact table is determined by the finest level in each dimension

Star schema Place fact table in center, with FK relationships to surrounding dimension tables. One-to-many relationship between dimensions and facts

Snowflake schema Where hierarchies are separated out into dimension tables.

14 Distributed DB

Terms

Distributed DB Single logical database spread physically across multiple computers in multiple locations connected by a communication link. Appears to users as if it is one database

Decentralized DB A collection of independent DB which are not networked as one logical db. Appears to users as if is several dbs

Vertical Scaling Increasing processing capabilities of existing servers to increase performance

Horizontal Adding more servers to increase performance

Objectives of distributed dbs

Location transparency User/Program does not need to know where the data is stored. Any request it makes from should be forwarded to the site related to its request.

Local autonomy A node can continue to function for local users if inter-connectivity to the network is lost. Admins can administer their local dbs individually with powers of: control local data, perform security checks, log transactions, recover local failures, full access to local data

Features of distributed dbs

- Locate data with a distributed catalog for metadata
- Determine location from with to retrieve data from and to process query components
- DBMS translation between nodes/servers with different local DBMSs using middlewares
- Data consistency through multiphase commit protocol
- Global primary key control
- Scalability
- Security, concurrency, query optimization, failure recovery

Advantages of distributed DBMS

- Good fit for geographically distributed organizations and users. Utilizes the internet
- Data often located near the site with the greatest demand
- Faster local data access (data specific to location)
- Faster data processing due to workload split amongst physical servers. A kind of horizontal scaling
- Allows modular growth in processing capabilities, by adding new servers as load increases through horizontal scaling
- Increased reliability and availability. Less danger of a single point of failure (SPOF) if the data is replicated
- Supports database recovery if data is replicated

Disadvantages of distributed DBMS

- Complexity of management and control. Because the DBMS must stitch together data across multiple sites.
- Complexity problems like: where is the current version of the record, who is waiting to update this information; how is this logic displayed to the application
- Data integrity, more exposure to improper updates
- Data integrity problems like: race condition of two users in different locations updating the same record. (solved with transaction manager, or master-slave database design)
- Security, for many servers implies higher chance of data breach. Requiring protection in networks and storage infrastructure against cyber and physical attacks
- Lack of standards, different protocols for different DDBMS
- Increased training and maintenance costs due to: complex IT infrastructure, increased disk storage, fast intra and inter network infrastructure, clustering software (that manages DDB)
- Increased storage required due to replication

Distribution options

Replication Duplicating data to different nodes. Most popular.

Partitioning Partitioning data into chunks stored in different nodes. A chunk can be a set of rows or a set of columns in a relation.

Horizontal partitioning Splits table rows across nodes

Vertical partitioning Splits table columns across nodes

Advantages of replication

- High reliability due to redundancy
- Faster access to data
- Avoid complicated distributed integrity routines, can simply refresh replicated data at scheduled intervals
- Decoupled/offline nodes don't affect data availability, transactions can continue even if some nodes are down
- Reduced network traffic if updates can be delayed

Disadvantages of replication

- Storage space, for all the copying
- Data integrity, high tolerance of applications for out of date data is needed
- Takes time for updates. Updates may cause performance issues for busy nodes for it needs to propagate. Retrieves incorrect data if updates have not propagated.
- High network communication capability is needed, for updates can place heavy demand on the networks. High speed network as expensive

Benefits of horizontal partitioning

- Data can be stored close to where it is used for efficiency, allows local access optimization
- Only relevant data is stored locally, better security
- Unioning across partitions is easier for querying

Problems of horizontal partitioning

- Inconsistent access speed when accessing data across partitions
- No data replication, SPOF vulnerability

Vertical partitioning has the same advantages and disadvantages as horizontal partitioning except combining data across partitions is more difficult for it needs joins instead of unions.

Tradeoffs in DDBMS

- Availability vs Consistency, either make data always available and partition tolerant or make it always consistent
- Synchronous vs Asynchronous updates, should changes be immediately visible everything but expensive, or later propagated but less expensive

The CAP theorem states that you can only have two out of three features for a distributed system

- Consistency, everyone sees the same data
- Availability, system stays up when nodes fail
- Partition tolerance, system stays up when network between systems fail

For synchronous updates

- Data is continuously kept up to date and all users will access the same data
- If a data item is updated anywhere on the network, the same update is immediately applied to all other copies of the data item or the update is aborted
- Ensures data integrity and minimizes complexity of knowing where the most recent version is located
- Slow response time and high network usage. Needing to spend time checking for successful update and propagation, committed update record must propagate to all servers

For async updates

- Must tolerate some temporary inconsistency, could be fine if the temp is short and well managed
- Acceptable response time, updates are applied locally and data replications are synchronized only in batches and predetermined intervals
- More complex to plan and design, to ensure data integrity and consistency

Some information system are suited for async updates (social media), others must need synchronous updates (finance systems).

15 NoSQL

NoSQL

- Most business data are tabular and suitable for relational model
- Adoption of NoSQL driven by problems of relational databases
- Polyglot persistence of different approaches to support different storage requirements
- Some aggregate data is not inherently tabular, costly to disassemble data and recompute aggregates. Can directly store as XML or JSON files.

Benefits of relational design

- Simple, can capture most business use cases
- Integrate/interface multiple applications via a shared data store
- Standard interface language SQL
- Adhoc queries, across and within table aggregates
- fast, reliable, concurrent, consistent

Problems of relational databases

- Object relational impedance mismatch, problem in translating object oriented data to relational data models (multilevel hierarchy, list of lists)
- Not good with big data
- Not good with clustered or replicated servers

Big data consists of three Vs

- Volume, larger quantity of data than typical for relation databases
- Variety, lots of different data types and formats
- Velocity, data comes at a very fast rate (from sensors or web clicks)

A data lake is a large integrated repository for internal and external data that does not follow a predefined schema (schema-on-read). It captures everything, and you can access the relevant parts later.

Conventions for schemas in databases

- Schemas on write, pre-existing data model, traditional relational databases. Traditional design process: requirement gathering, formal data modeling, database schema, usage based on predefined schemas
- Schema on read, data model determined later when reading, depends on how it is used. Big data design process: collect large amounts of data with local structures, stores data in data lake, analyze the stored data and create meaningful structures, structuring and organize the data after data analysis

Schema-on-read problems with: overhead when reading for different data formats, difficult for analysis since no way ahead of time to know relevant data fields and need to filter for it, takes more space storing schemas.

The features and purpose of a NoSQL database are

- doesnt use relational model nor SQL
- runs well on distributed servers
- open-source
- modern, built for the modern web
- schema-less (could have implicit schema)
- support schema on read
- not ACID compliant
- BASE
- Purpose: Improve programmer productivity in avoiding OR mismatch
- Purpose: Handle larger data volumes and throughput (big data)

Types of NoSQL databases

Key Value Keys are strings. Values are any datatype — the application is in charge of the interpretation. Operations are put for storing, get to fetch, and update to update.

Document based Similar to a key-value store except that the document (generalized key value pair) is examinable by the database and is structured, so we can query parts its contents and update parts of it. A document generally is a JSON file denoting one entry in a table.

Column Family It is a generalized key-value store. Each row is represented as a row key plus a dictionary of key value fields representing the columns. The columns of each row can differ in names, structure, and format. The database will then store similar columns together on disk (ordered by columns instead of by rows), related columns are grouped into families. This makes analysis faster as less data is fetched for aggregating data across columns, due to disk groupings of relevant data (queries often don't require all columns). Like automatic vertical partitioning.

Graph Databases A graph is a node and arc network (a network of nodes and arcs). These graphs are difficult to design in relational databases. A graph database is designed to store such graphs with entities and relationships. It facilitates fast graph queries (like finding extended friends) which deduce knowledge from the graph.

Aggregate oriented databases

Definition Key-value, document store, column family. Store business objects in aggregate form

Benefits Entire aggregate of data is stored together, no need for transactions in updating or reading

Efficient storage on clusters and distributed databases

Problems Hard to analyze across subfields of aggregates (aggregate on sales, hard to compute aggregate on products)

Fowler's version of CAP theorem states that for distributed databases, under a data partition, we must choose between availability of consistency. Different types of data in the same system can require different availability and consistency levels.

BASE Principles

Basically Available The constraint that the system does guarantee availability of data in responding to all requests, but the data may be in an inconsistent state or changing state.

Soft state The state of the system could change over time even where there are no inputs. This is due to changes from the "eventually consistency" clause.

Eventual consistency The system will eventually become consistent once it stops receiving inputs. The data will eventually propagate. But the system can always continue receiving inputs and not check the consistency of any transactions before processing the next one.

Variations of the eventual consistency principle. In practice, a number of theses are fulfilled (read-your-write and monotonic reads)

Casual Processes that has casual relationships will see consistent data

Read your write Process will always access the most recent data after it updates it and will not see an older value

Session Within a session, guarantee read-your-write

Montonic read if a process has seen a particular value, all subsequent reading of this data by this process will not return any previous values

Montonic writes Guarantees serialize writes by the same process

16 Spatial DB

Special data types

- Regular data types can be processed using basic operations
- Special data types are harder to process, retrieve, or operate with
- Includes images, videos, audio
- Requires indexes to process queries efficiently

Spatial data

- Contains items located in space
- Requires complex computation for retrievals like: intersection, range queries, nearest neighbor
- No trivial ways to sort them

Quadtrees

- Indexing spatial data in 2d space can exponentially reduce the number of comparisons using the Quadtree index.
- These are implemented in Oracle Spatial and other DBMSs as extensions.
- The quadtree has an implicit tree structure on quads that divides a space, with fast implementations of the nearest neighbor algorithm using priority queue
- Each node is associated with a rectangular region of space. The root node is the entire target space
- Each division is due to a rule based on the specific data type. This division is recursive until a stopping conditioin
- Non-leaf nodes divides its region into four equally sized quads, containing four child nodes corresponding to them.
- Leaf nodes have between zero to some fixed maximum number of points

Kd tree is another data structure used to index in multiple dimensions.

- Each level of a kd tree partitions the space into two against one dimension
- We choose a different dimension to partition the space between levels, cycling through the dimensions across levels
- Aim to partition so that about half the points in the subtree will fall on either side
- Recursively partition until a node has less than a given number of points

R-trees

- Both quadtrees and kd trees are only faster in memory. Some databases also need a tree index on disk.
- A generalized n-dimensional B+ tree. It is used to index sets of rectangles or polygonal data.
- It stores only the nodes in memory, while keeping the last few nodes and objects on disk.
- Well-supported across DBMS. Variants are R+ or R* trees.
- Generalize the one dimensional interval coverage within each B+ node to n-dimensional interval coverage (rectangles) of each R-tree node
- Often used for the common $N = 2$ case, generalization is simple, but they only work well with small N data
- For each R-tree node, we partition the set of polygons inside it into smaller sets of new nodes, creating smaller r-tree nodes each having a bounding box
- The clusters are determined based on a rule (proximity), and is done recursively until a stopping condition
- Define a bounding box of a node as the minimum sized rectangle that contains all polygons in the node.
- The disadvantages are that bounding boxes of children of a node may overlap with others.

To find data items intersecting a given region

- Start from the root node
- If node is a leaf, output the data items whose keys intersect the region
- For each child of the current node, check if their bounding box intersects the region, if so then recursively search
- This can very inefficient in the worst case since multiple paths will be searched due to the overlaps. But it works fine in practice.

To do nearest neighbor from a point

- Start by adding the root node to the priority queue
- Pull element from queue repeatedly.
- If the node is not an element, push its subnodes onto the queue. If the subnodes are nodes with bounding boxes, use the closest point of the box to the point, else use the closet point on the polygon to the point.
- If the node is an element, that is the nearest neighbor
- Repeat until we found the kth nearest neighbor

There is no point going through all index types for all data types. What we should do is instead to

- For a given data set, during the DBMS uploading stage
- Find the potential queries
- Research indices the DBMS has for this data type
- Research which queries that can be done more efficiently on which index
- Create index if we have large data
- Monitor, tune or create other indices

There are also special algorithms that the DBMS implements for special data types and queries.

- Some algorithms are: shortest path, tsp.
- We can run these algorithms using extensions to SQL. It can also extend the relational model to incorporate new data types as tables. These can be new functions to filter based on distance, or keywords indicating special data type tables.
- But these may not be enough, so we have new technologies on storage and querying for some specific data types (big data and NoSQL).