- **Tuples and Relations:**
  - *Tuple:* a *k-tuple* is an ordered sequence of k values
  - If $D_1, D_2, \ldots, D_k$ are sets of elements then the cartesian product $D_1 \times D_2 \times \ldots D_k$ is the set of all k-tuples $(d_1, d_2, \ldots, d_k)$ such that $\forall 1 \le i \le k : d_i \in D_i$
  - *Relation:*
    - A *k-ary relation* is a subset of $D_1 \times D_2 \times \ldots D_k$ where each D_i is a set of elements
    - D_i is the *domain (or datatype)* of the i^{th} column of the relation
    - Domains may be enumerated $\{``AMS'', ``CMPS'', ``TIM''\}$ or may be of standard types
  - An *attribute* is the name of a column in a relation
  - A *relation schema* R is a set $\{A_1, \ldots, A_k\}$ of attributes written $R(A_1, \ldots, A_k)$, where A_i is the name of the i^{th} column.
  - A *relation database schema* or *database schema* is a set of schemas with disjoint relation names.
- **SQL Primitives:**
  - CHAR(n): fixed-length string of up to n characters (blank-padded with trailing spaces)
  - VARCHAR(n): also a string of up to n characters
  - BIT(n): padded on the right with 0s.
  - BIG VARYING(n): works like VARCHAR
  - BOOLEAN: true, false, unknown
  - INT or INTEGER: works like in C
  - SHORTINT: works like short int
  - DECIMAL(n, d), NUMERIC(n, d): total of n digits, d of them to the right of the decimal point
  - FLOAT(p), FLOAT, REAL
  - DOUBLE PRECISION: analagous to double in c
  - DATE, TIME, TIMESTAMP, INTERVAL: constants are character strings of specific form e.g. DATE '2017-09-13'
    - TIME A - TIME B results in an INTERVAL
    - TIME A + INTERVAL B results in a TIME
    - Similarly for TIMESTAMP and DATE
- **Tables:**
  - A *key* of a relation schema R is a subset K of the attributes of R such that:
    - $\forall Y \in R : K \to Y$
    - No proper subset of K has the above property
  - A *superkey* is a set of attributes that includes a key
  - CREATE TABLE R(A, B, C, PRIMARY KEY(A)):
    - None of the tuples in R can have null A values
    - Rows are uniquely identified by their A values
    - There can be ≤ 1 primary key for a table
  - CREATE TABLE S(D, E, F, UNIQUE(D)):
    - Rows in S can contain null D values
    - Rows with *non-null* D values are uniquely identified by their D values
    - There can be multiple unique constraints in addition to a primary key
  - CREATE TABLE T(G NOT NULL, H DEFAULT 'foo'):
    - If no default value is specified and no value is entered then the value will be NULL
    - NOT NULL prevents null values
- **Queries:**
  - Basic form:
```
SELECT [DISTINCT] c1, c2, ..., cm
FROM R1, R2, ..., Rn
[WHERE condition]
```
  - SELECT:
    - Projection: SELECT title, year
    - DISTINCT: Removes duplicate tuples from result
    - Aliasing: SELECT title AS name
    - Expressions are allowed in the SELECT clause. Ex: SELECT title AS name, length * 60 AS durationInSeconds
    - Constants can also be included: SELECT title AS name, length * 60 AS durationInSeconds, 'seconds'AS inSeconds
  - WHERE:
    - Comparison operators: =, <>, <, >, <=, >=
    - Logical connectives: AND, OR, NOT
    - Arithmetic expressions: +, -, *, /, etc
    - In general the WHERE clause is a boolean expression where each condition is of the form *expression op expression*
  - Pattern matching with the LIKE operator:
    - s LIKE p, s NOT LIKE p
    - s is a string, p is a pattern
    - '\%' stands for 0 or more arbitrary characters
    - '_' stands for exactly one arbitrary character
    - quotes: WHERE x LIKE '''' matches one
    - quotes: WHERE x LIKE '''''' matches two
    - \% or _: WHERE x LIKE '!\%\%!'ESCAPE '!' where ! can be any character
  - DATE and TIME and TIMESTAMP
    - Separate data types
    - Constants are character strings of the form:
```
DATE '2015-01-13'
TIME '16:45:33'
TIMESTAMP '2015-01-13 16:45:33'
```
    - DATE, TIME, TIMESTAMP can be compared using ordinary comparison operators e.g. WHERE ReleaseDate <= DATE '1990-06-19'
  - If Salary is NULL then the following will be UNKNOWN:
    - Salary = 10
    - Salary <> 10
    - 90 > Salary OR 90 <= Salary
    - Salary = NULL
    - Salary <> NULL
  - Ordering the result:
    - ORDER BY presents the result in a sorted order
    - By default the result will be ordered in ascending order ASC
    - For descending order on an attribute you write DESC in the list of attributes
  - Multiple relations in FROM clause: for every tuple $t_1 \in R_1, t_2 \in R_2, \ldots, t_n fromR_n$ if $t_1, \ldots, t_n$ satisfy *condition* then add the resulting tuple that consists of $c_1, c_2, \ldots, c_m$ components of t into the result
- **Joins:** With relations R(A,B,C) and S(C,D,E)
  - R JOIN S ON R.B=S.D AND R.A=S.E:
    - Selects only tuples from R and S where R.B=S.D and R.A=S.E
    - Schema of the resulting relation: (R.A, R.B, R.C, S.C, S.D, S.E)
    - Equivalent to:
```
SELECT *
FROM R, S
WHERE R.B=S.D AND R.A=S.E;
```
  - R CROSS JOIN S:
    - Product of the two relations R and S
    - Schema of the resulting relation: (R.A, R.B, R.C, S.C, S.D, S.E)
    - Equivalent to:
```
SELECT *
FROM R, S;
```
  - R NATURAL JOIN S:
    - Schema of the resulting relation: (A, B, C, D, E)
    - Equivalent to:
```
SELECT R.A, R.B, R.C, S.D, S.E
FROM R, S
WHERE R.C = S.C
```
- **Set and Bag Operations:** R(A,B,C), S(A,B,C)
  - UNION: Set union
    - Input to union must be *union-compatible*: R and S must have attributes of the same type, in the same order
    - Output of the union has the same schema as R or S
    - Meaning: Output consists of the *set* of all tuples from R and from S
    - Could have been called UNION DISTINCT
```
(SELECT * FROM R)
UNION
(SELECT * FROM S)
```
  - UNION ALL: Bag union
    - Input must be *union-compatible*
    - Output has the same schema as R or S
    - Output consists of the collection of all tuples from R and from S *including duplicates.*
    - Attributes/column names may be different - R's are used
  - INTERSECT, INTERSECT ALL: set/bag intersection
    - Input must be union-compatible.
    - $Query_1$ INTERSECT $Query_2$
    - $Query_1$ INTERSECT ALL $Query_2$
    - Find all tuples that are in the results of both $Query_1$ and $Query_2$.
    - INTERSECT is distinct. INTERSECT ALL reports duplicates.
  - EXCEPT, EXCEPT ALL: set difference, bag difference
    - Must be union-compatible
    - $Query_1$ EXCEPT $Query_2$
    - $Query_1$ EXCEPT ALL $Query_2$
    - Find all tuples that are in the result of $Query_1$ and not in the result of $Query_2$
    - EXCEPT is distinct, EXCEPT ALL is not
  - Order of operations: INTERSECT has higher precedence than UNION and EXCEPT.
- **Subqueries:**
  - A query embedded in another query
  - Can be used as a boolean or can return a constant or can return a relation
  - IN, NOT IN: used to select from subquery that returns relation
  - WHERE A < ANY: checks that attribute A is less than at least one of the answers returned by the subquery.
  - EXISTS: Checks that subquery returns non-empty result. Also: NOT EXISTS
- **Aggregates and Grouping:**
  - Basic SQL has 5 aggregation operators: SUM, AVG, MIN, MAX, COUNT
  - Aggregation operators work on scalar values, except for COUNT(*) which counts the number of tuples
  - GROUP BY clause follows the WHERE clause
    - Let Result begin as an empty multiset of tuples
    - For every tuple t_1 from R_1, t_2 from $R_2, \ldots, t_n$ from R_n: if $t_1, \ldots, t_n$ satisfy *condition* then add the resulting tuple that consist of $c_1, c_2, \ldots, c_m$ of the t_i into Result
    - Group the tuples according to the grouping attributes - if GROUP BY is omitted, the entire table is one group
  - NULLs are ignored in any aggregation
    - They do not contribute to the SUM, AVG, COUNT, MIN, MAX of an attribute
    - COUNT(*) = the number of tuples in a relation even if some columns are NULL
    - COUNT(A) is the number of tuples with *non*-NULL values for A
    - SUM, AVG, MIN, MAX on an empty result (no tuples) is NULL
    - COUNT of an empty result is 0
    - GROUP BY does *not* ignore NULL
  - HAVING clause:
    - Choose groups based on some aggregate property of the group itself
    - Same attributes and aggregates that can appear in the SELECT can appear in the HAVING clause condition
    - Can use EVERY to constrain HAVING to all tuples in the group e.g. HAVING COUNT(*)> 1 AND EVERY (S.age <= 40)
- **Database Modification Statements:**
  - INSERT INTO R(A1, ..., An)VALUES (v1, ..., vn): a tuple $(v_1, \ldots, v_n)$ is inserted into R such that $A_i = v_i \forall i$ and default values (perhaps NULL) are entered for any missing attributes.
  - DELETE FROM R WHERE <condition>: Deletes *all* tuples such that the condition evaluates as true - if there is no WHERE clause it will delete all tuples in R
  - UPDATE R SET <new-value-assignments> WHERE <condition>: Change the given attribute to the new value in every tuple in R where the condition is true
  - Semantics: database modifications are completely evaluated on the old state of the database producing a new state of the database
- **Transaction:**
  - Transactions provide ACID properties: atomicity, consistency, isolation, durability
  - START TRANSACTION or BEGIN TRANSACTION: marks the beginning of a transaction, followed by one or more SQL statements
  - COMMIT: Ends the transaction. All changes are durably written to the backing store and become visible to other transactions.
  - ROLLBACK: Causes the transaction to abort or terminate. None of the changes are committed.
  - SET TRANSACTION READ ONLY:
    - set *before* the transaction begins, tells the SQL system that the next transaction is read-only
    - SQL uses this to parallelize many read-only transactions
  - SET TRANSACTION READ WRITE:
    - Tells SQL that the next transaction may write data in addition to read
    - Default option if not specified, often not specified
  - Dirty Reads: *Dirty data* refers to data that is written by a transaction but has not yet been committed by the transaction
  - Isolation levels:
    - SET TRANSACTION READ WRITE ISOLATION LEVEL READ UNCOMMITTED
    - Default isolation level depends on system, most run with READ COMMITTED or SNAPSHOT ISOLATION
    - READ COMMITTED: only clean(committed) reads but you might read data committed by other transactions
    - REPEATABLE READ: repeated queries of a tuple during a transaction will retrieve the same value. Also, a second scan may return 'phantoms' which are tuples newly inserted while the transaction is running.
    - SERIALIZABLE: Can be replayed one by one.
- **Constraints:**
  - Key/Unique constraint: No repetitions of this value in the relation
  - Foreign-key constraint: Referential integrity. Value must exist in another relation as specified. Referenced attributes must be PRIMARY KEY or UNIQUE.
    - With attribute:
      beer CHAR(20)REFERENCES Beers(name)
    - As schema element:
```
FOREIGN KEY (beer)
  REFERENCES Beers(name)
```
  - Value-based constraint: CHECK(<condition>). Checked on insert or update.
  - Enforcing constraints:
    - *Default:* Reject the modification
    - *Cascade:* Make changes to maintain consistency
    - *Set NULL:* Change dependent values to NULL
    - Selected independently with ON [UPDATE, DELETE] [SET NULL, CASCADE]
  - Assertions: database-schema elements, like relations or views. Defined by:
```
CREATE ASSERTION <name>
  CHECK (<condition>)
```
- **Triggers:**
  - Event-Condition-Action or *ECA* rule:
    - *Event:* Typically a DB modification
    - *Condition:* Any SQL boolean expression
    - *Action:* Any SQL statements
  - *Create:*
    - CREATE TRIGGER <name>
    - CREATE OR REPLACE TRIGGER <name>
  - *The Event:*
    - [AFTER, BEFORE] [INSERT, DELETE, UPDATE [ON]]
    - Can be INSTEAD OF if the relation is a view
  - FOR EACH ROW
    - Triggers are 'row-level' or 'statement-level'

    - FOR EACH ROW means row-level, absence means statement-level
    - *Row level triggers* execute once for each tuple
    - *Statement-level* execute once for a SQL statement
  - REFERENCING
    - INSERT statements imply a new tuple or table
    - DELETE implies an old tuple or table
    - UPDATE implies both
    - [NEW OLD] [TUPLE, TABLE] AS <name>
  - *The Condition:*
    - Any boolean condition
    - Evaluated on the DB as it existed BEFORE/AFTER the event
  - *The Action:*
    - There can be more than one SQL statement
    - Surround with BEGIN ... END
- Relational Algebra:
  - Selection: $\sigma_{condition}(R)$
    - *Input:* Relation with schema $R(A_1, \dots, A_n)$
    - *Output:* Relation with attributes $A_1, \dots, A_n$
    - *Meaning:* Extracts rows which satisfy condition
  - Projection: $\pi_{<attribute\ list>}(R)$
    - *Input:* Relation with schema $R(A_1, \dots, A_n)$
    - *Output:* Relation with listed attributes
    - *Meaning:* Extracts all rows from R and outputs only those attributes listed
  - Union: $R \cup S$
    - *Input:* Union-compatible relations R, S
    - *Output:* Relation with the same type of R (or S)
    - *Meaning:* The output is the set of all tuples in either R or S or both
    - Both *commutative* and *associative*
  - Set difference: $R - S$
    - *Input:* Union-compatible relations R, S
    - *Output:* Relation with the same type of R (or S)
    - *Meaning:* $\{x \mid x \in R,\ x \notin S\}$
  - Cross-product: $R \times S$
    - *Input:* $R(A_1, \dots, A_n), S(B_1, \dots, B_m)$
    - *Output:* $T(A_1, \dots, A_n, B_1, \dots, B_m)$
    - *Meaning:* $R \times S = \{(a_1, \dots, a_n, b_1, \dots, b_m) \mid (a_1, \dots, a_m) \in R,\ (b_1, \dots, b_m) \in S)\}$
  - Intersection: $R \cap S$ is a *derived operator* $R \cap S = R - (R - S) = S - (S - R)$
  - Renaming: $\rho_{S(A_1, \dots, A_n)}(R)$
    - *Input:* A relation R and a set of attributes $\{B_1, \dots, B_n\}$
    - *Output:* A relation S and attributes $A_1, \dots, A_n$
  - Natural Join: $R \bowtie S$
    - *Input:* Two relations R and S where $\{A_1, \dots, A_k\}$ is a set of common attributes between them
    - *Output:* A relation where its attributes are $attr(R) \cup attr(S)$. Consists of $R \times S$ without any repeats of the common attributes.
    - *Meaning:* $R \bowtie S = \pi_{(attr(R) \cup attr(S))}(\sigma_C(R \times S))$ (where $C = R.A_1 = S.A_1$ AND ... AND $R.A_k = S.A_k$)
  - Semi-Join: $R \ltimes S = \pi_{attr(R)}(R \bowtie S)$
  - Theta Join: $\bowtie_\theta$
    - *Input:* $R(A_1, \dots, A_n), S(B_1, \dots, B_m)$
    - *Output:* $T(A_1, \dots, A_n, B_1, \dots, B_m)$ Identical attributes are disambiguated with the relation names.
    - *Meaning:* Equivalent to writing $\sigma_\theta(R \times S)$.
  - Division: R/S or $R \div S$
    - *Input:* Two relations R and S such that $attr(S) \subset attr(R)$ and $attr(S) \neq \emptyset$
    - *Output:* Relation whose attributes are in $attr(R) - attr(S)$
    - *Meaning:* Given $R(a, b, c, d)$, $S(b, d)$: $R \div S$ outputs (a, c) for each tuple in S such that $R.b = S.b$ and $R.d = S.d$
  - *Independence:* The five basic operators are indepdent of each other.
    - $\times$ increases columns
    - $\cup$ increases rows
    - π decreases columns
    - σ is binary, decreases rows
    - $-$ is unary, decreases rows
- STORED PROCEDURES
  - PSM or *persistent stored modules* store procedures as DB schema elements
  - Basic form:
```
CREATE PROCEDURE <name> (
  <parameter list>)RETURNS <type>
  <optional local declarations>
  <body>;
```
  - Uses mode-name-type triples where the mode can be:
    - IN: procedure uses value, doesn't change
    - OUT: procedure changes value, doesn't use
    - INOUT: both
  - Function parameters must be of type IN
  - Procedures invoked with CALL <name>(<parameters>)
  - May be used in SQL expressions wherever their return type fits
  - RETURN <expression> sets the return value but doesn't terminate execution
  - DECLARE <name> <type> declares local variable
  - BEGIN ... END for groups of statements
  - SET <variable> = <expression>
  - IF <condition> THEN <statements> END IF;
  - Add ELSE <statement(s)> as IF...THEN...ELSE...END IF
  - Add additional cases: IF...THEN...ELSEIF...ELSE
  - Basic loop:
```
<loop name>: LOOP
<statements>
END LOOP;
```
  - Leave loop with LEAVE <loop name>
  - Also: WHILE <condition> DO <statements> END WHILE;
  - Also: REPEAT <statements> UNTIL <condition> END REPEAT;
  - Queries producing one value can be the expression in an assignment.
  - Queries returning one row: SELECT ... INTO ...
  - Cursors:
    - DECLARE c CURSOR FOR <query> to declare, binds values
    - OPEN c to open
    - CLOSE c to close
    - FETCH FROM c INTO x1, x2, ..., xn sets the x's to the values of a tuple
    - c moves to the next tuple automatically
    - DECLARE NotFound CONDITION FOR SQLSTATE '02000'
    - CURRENT OF c allows use in WHERE for current tuple
- EMBEDDED SQL
  - Uses preprocessor in host language to turn SQL statements into host library calls
  - Declare shared variables with
```
EXEC SQL BEGIN DECLARE SECTION;
<host-language declarations>
EXEC SQL END DECLARE SECTION;
```
  - Shared variables must be preceded by a colon. Can be used as if they were constants provided by the host language program. Can get values from SQL statements and pass those values to the host language program. In the host language they behave like any other variable.
  - Usage: <SQL varname> = :<host varname>
  - Every SQL statement must begin EXEC SQL ...
  - PrepareStatement:
```
EXEC SQL PREPARE <query-name>
  FROM <text of query>;
```
  - Execute: EXEC SQL EXECUTE <query-name>;
  - If only using it once: EXEC SQL EXECUTE IMMEDIATE <text>;
- JDBC
  - Provides Statement and PreparedStatement classes
  - Usage:
```
import java.sql.*;
Class.forName(com.mysql.jdbc.Driver);
Connection myCon = DriverManager.getConnection(...);
```
  - createStatement() takes a string of a SQL statement
  - prepareStatement() is configurable
  - Use executeQuery or executeUpdate as appropriate
  - Both return ResultSet which uses something like a cursor, rs.next() gets the next one
  - When a resultSet returns a tuple we get the components with getX(i) where X is a type and i is the component number
- FUNCTIONAL DEPENDENCIES
  - Let R be a relation schema. An FD is an integrity constraint $X \to Y$ where X and Y are non-empty subsets of the attributes of R
  - A relation instance r of R *satisfies* the FD $X \to Y$ iff for every pair of tuples $t, t' \in r : t[X] = t'[X] \implies t[Y] = t'[Y]$
  - A set $\mathcal{F}$ of FDs implies an FD F if for every instance r that satisfies $\mathcal{F}$, it must be true that r satisfies F: $\mathcal{F} \models F$
  - ARMSTRONG'S AXIOMS
    - *Reflexivity:* $Y \subseteq X \implies X \to Y$ (trivial)
    - *Augmentation:* $X \to Y \implies \forall Z:\ XZ \to YZ$
    - *Transitivity:* $X \to Y,\ Y \to Z \implies X \to Z$
  - $\mathcal{F} \vdash F$ means F can be derived from $\mathcal{F}$ using Armstrong's Axoms ($\mathcal{F}$ generates F)
  - Closure: $\mathcal{F}^+$ means the set of all FDs implied by $\mathcal{F}$
  - $\mathcal{F} \vdash X \to A \iff A \in X^+$
- NORMAL FORMS
  - *First Normal Form (1NF):* Very basic requirement of the data model. Type of every attribute must be atomic.
  - *Boyce-Codd Normal Form (BCNF):* Let R be a relation schema, $\mathcal{F}$ a set of FDs that holds for R, A an attribute in R, and X a subset of attributes in R. R is in $BCNF$ iff for every FD $X \to A$: $A \in X$, or X is a superkey.
  - *Third Normal Form (3NF):* Like $BCNF$ except A can be a part of a key of R.
- DECOMPOSITION
  - Given a relation R, a decomposition is a set of attributes $X_1, \dots, X_k$ (not necessarily disjoint) such that each $X_i \subseteq attr(R)$ and $X_1 \cup \dots \cup X_k = attr(R)$
  - $\pi_{X_i}(R) = R_i$ with instances of R written r and instances of R_i written r_i
  - Used to eliminate anomalies
  - *Lossless Join Decomposition* if it allows recreation of R by using $r_1 \bowtie \dots \bowtie r_k$
  - Lossless decomposition of R into $R_1(X_1), R_2(X_2)$ requires $\mathcal{F}^+$ has either $X_1 \cap X_2 \to X_1$ or $X_1 \cap X_2 \to X_2$.
  - The FDs on R_i are the FDs in $\mathcal{F}^+$ that mention only $attr(R)$
  - The decomposition is *dependency-preserving* if, when the R_i's are rejoined, the FDs that were on the R_is imply all of the original FDs in $\mathcal{F}$
  - Given a schema and set of FDs, it's always possible to decompose into a set of
    - $BCNF$ relations which eliminate anomalies and is a lossless join decomposition but might not be dependency-preserving
    - $3NF$ relations which is a lossless join decomposition and is dependency-preserving but might not always eliminate anomalies
- SEMI-STRUCTURED DATA
  - XML
    - *Tags* come in pairs <data>...</date>
    - Must be properly nested
    - Start the doc with <?xml ... ?>
    - Normal declaration is <?xml version = "1.0"standalone = "yes"?>
    - An attribute is anything surrounded by a tag
    - A list is multiple tags used repeatedly
    - A tag attribute goes inside the tag e.g. <word lang="en">
    - Can use id="foo" attribute to specify an element and then reference it with idrefs="foo"
    - DTD specifies syntax for a document. DTD structure:
```
<!DOCTYPE <root tag> [
  <!ELEMENT <name>(<components>)>
]>
```
    - Uses regex ?*()|+,
    - (#PCDATA) means *parsed character data* e.g. text
    - #IMPLIED means optional, otherwise #REQUIRED
    - Use ID and IDREF(S) in DTD to enforce constraints
    - Use SYSTEM "foo" in doctype tag to specify source for dtd
  - JSON
    - Unordered sets of name/value pairs
    - Normal: {a:b, c:d}
    - Arrays are ordered: [a,b,c,]
    - Strings wrapped in double quotes support backslash escapes
    - Numbers: integer, real, scientific
- STAR SCHEMA
  - Common organization for OLAP data in a warehouse
  - DIMENSION TABLE
    - Smaller, largely static info describing the data that's in the Fact
    -
  - FACT TABLE
    - Large accumulation of facts such as sales
    - The key consists of values from dimension tables (foreign keys)
    - Facts usually 'insert mostly' with rare updates
    - *Dimension Attributes:* The key of a dimension table, references Dimenson Tables
    - *Dependent Attributes:* A fact value determined by the dimension attributes
  - *Roll-up* means aggregate along one or more dimensions
  - *Drill-Down* means 'dis-aggregate' or break an aggregate into its constituent parts
  - R [LEFT, RIGHT, FULL] OUTER JOIN S [ON <condition>]:
    - Used to join R and S where there exists some tuple in R which has no match in S
    - Fills in missing attributes wiht NULL
    - LEFT means pad dangling tuples of R only (and vice versa)
    - FULL means pad both, default
  - Taking the cartesian product of the dimension table keys, then taking LEFT OUTER JOIN of that with the fact table, will give you entries for every combination of dimensions (not just those that have entries in the fact table)
  - COALESCE(x, 0) has value x if x isn't NULL, otherwise 0.
- ACID
  - ATOMICITY: An atomic transaction happens as one unit, either the whole thing commits or none of it does.
  - CONSISTENCY: A consistent transaction brings the DB from one valid state to another valid state with respect to any constraints.
  - ISOLATION: Concurrent isolated transactions would have the same result if run sequentially.
  - DURABILITY: A committed transaction will remain committed even in the event of a hardware failure.